UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

INE5430 - INTELIGÊNCIA ARTIFICIAL

Relatório - Gato vs Não-Gato

Eduardo Borges Siqueira

Rafael Moresco Vieira

Samuel Moreira Ransolin

Florianópolis

2023

Sumário

1. Introdução

O reconhecimento de padrões é uma tarefa fundamental nas áreas de aprendizado de máquina e inteligência artificial (*IA*). Ela consiste em treinar um modelo para identificar e classificar corretamente conjuntos de padrões de entrada. Uma abordagem comumente utilizada nesse processo é o uso de redes neurais artificiais (*RNA*), que são capazes de aprender e generalizar a partir dos dados fornecidos.

Neste trabalho, o objetivo principal é treinar e testar uma inteligência artificial para a classificação de imagens de gatos. As imagens são representadas por matrizes de tamanho 64x64x3, ou seja, uma imagem quadrada, com 64 *pixels* de altura, 64 *pixels* de largura e 3 camadas de cores, onde cada valor dessa matriz representa a intensidade de sua respectiva camada de cor: vermelho, verde ou azul. O objetivo é desenvolver um modelo capaz de receber essas imagens como entrada e atribuir à classe correta: "Gato" ou "Não Gato".

O trabalho propõe a resolução do problema utilizando duas diferentes abordagens. A primeira delas é utilizar um modelo de regressão logística (um *perceptron*), que atuará como um modelo de rede neural de camada única. A segunda é utilizar uma rede neural de camada rasa, que consiste em uma estrutura com uma camada de entrada, uma camada intermediária e uma camada de saída.

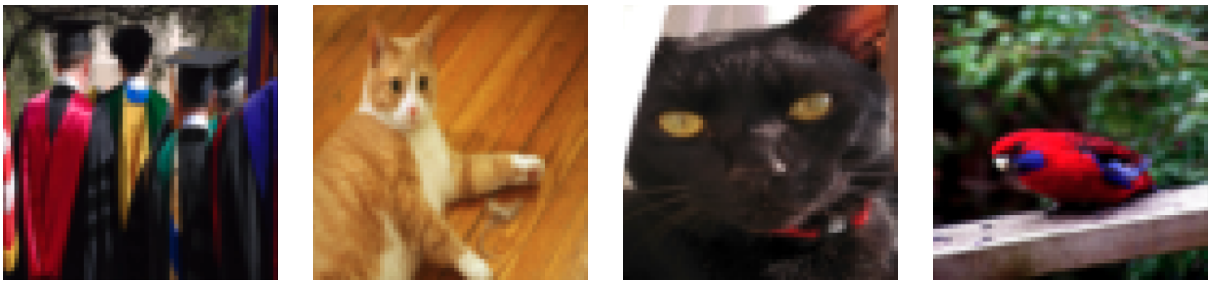
2. Desenvolvimento

Para desenvolvimento dos modelos, foi utilizada a linguagem de programação Python com versão superior à 3.8, utilizando bibliotecas de apoio e *frameworks* para criação e gerenciamento dos modelos, das quais, destacam-se:

- ***h5py*** para gerenciamento de arquivos *.h5*;
- ***matplotlib*** para visualização de gráficos;
- ***numpy*** para manipulação dos dados extraídos dos *datasets*;
- ***scikit-learn*** para regressão logística;
- ***tensorflow*/*keras*** para a rede neural de camada rasa.

Os *datasets* para treino e teste dos modelos foram disponibilizados pelo professor da disciplina em dois arquivos *h5* de mesma estrutura, em uma chave contendo as imagens e em outra suas respectivas classificações (0 para não gato, 1 para gato). Ao total, os arquivos comportam 209 imagens para treinamento e 50 imagens para validação.

Exemplos de imagens dos datasets:



2.1. Metodologia

Para a realização do trabalho, foram criados três arquivos de código: um para a implementação do modelo de regressão logística, outro para a implementação do modelo de rede neural de camada rasa e um último de suporte para comportar funções de uso comum aos dois módulos, como a extração dos *datasets* e normalização dos dados.

Em geral, o código dos modelos de *IA* discorre da seguinte forma: os *datasets* de treino e validação são extraídos em diferentes *arrays*, em que para cada conjunto de dados passa a existir uma estrutura que comporta as características de dados de entrada (um *array* multidimensional de imagens e seus pixels, como explicado previamente) e outra que comporta sua classificação (um *array* de *labels* que classifica sua imagem em gato ou não gato). Então, os dados das imagens são pré-processados e normalizados, um modelo de classificação é construído, testado com os dados de treinamento, e avaliado com os dados de teste. Assim, resultados e métricas relevantes, como acurácia e matriz de confusão, são apresentados ao usuário. Um breve detalhamento para cada abordagem está descrito abaixo.

2.1.1. Regressão Logística

Após a extração, os valores dos pixels das imagens, que estão em formato multidimensional (altura x largura x camada de cores *RGB*), são comprimidos em um *array* unidimensional. Esses valores, que variam no intervalo de 0 a 255, são divididos por 255 para que variem em um novo intervalo de 0 a 1, normalizando-os.

A classe *LogisticRegression* da biblioteca *scikit-learn* foi utilizada para criar os modelos de regressão logística. Por fins de comparação, foram utilizados três modelos com diferentes algoritmos de minimização de custo (*L-BFGS* e *LIBLINEAR*) e termos de regularização distintos (nenhum, *L1* e *L2*). Vale ressaltar que a função de custo utilizada é implementação interna da biblioteca, e parâmetros como épocas de treino e taxa de aprendizado não precisaram ser definidos.

O modelo é treinado com os dados de treinamento normalizados, e a acurácia e matriz de confusão para os dados de treinamento e teste são exibidos ao usuário.

2.1.2. Rede de Camada Rasa

De forma equivalente à abordagem para a regressão logística, os *datasets* de treinamento e teste são extraídos, mas dessa vez são apenas normalizados. Isso porque o pré-processamento das imagens passa a ser responsabilidade da *API Sequential* da biblioteca *Tensorflow/Keras*, utilizada agora para a modelagem da rede neural de camada rasa.

Essa rede neural é modelada com uma camada de entrada, uma camada intermediária e uma camada de saída. A camada de entrada, do tipo *Flatten*, recebe e pré-processa as imagens, achatando o *array* multidimensional de pixels. Para a camada escondida, uma camada densa de neurônios, diferentes números de neurônios (60, 200 e 1000) e funções de ativação (tangente hiperbólica, ReLU e sigmóide) foram utilizadas para determinar o melhor modelo, com os pesos e vieses inicializando seguindo uma distribuição uniformemente randômica. A camada de saída é composta por um neurônio, utilizando a função de ativação sigmóide, já que a classificação é binária.

Para o treinamento da rede, utilizou-se a função de custo de entropia cruzada binária, otimizada pelo algoritmo de gradiente descendente estocástico, variando a taxa de aprendizado e número de épocas de treino.

3. Testes e Resultados

Em geral, a utilização das bibliotecas simplificou muito o trabalho de treino, testes e coleta de resultados. As métricas utilizadas para avaliar os modelos foram acurácia, valor da função de custo e matriz de confusão para os dados de treinamento e teste.

Seguem os resultados para os modelos desenvolvidos.

3.1. Regressão Logística

- Solver: *L-BFGS*; Regularizador: nenhum.

```
================================================
Regulaziração: None
========= Estatísticas de treinamento =========
Acurácia: 1.0
Matriz de confusão:
 [[1. 0.]
 [0. 1.]]
========== Estatísticas de validação ==========
Acurácia:  0.68
Matriz de confusão:
 [[0.64705882 0.35294118]
 [0.3030303  0.6969697 ]]
================================================
```

- Solver: *LIBLINEAR*; Regularizador: *L1*.

```
================================================
Regulaziração: l1
========= Estatísticas de treinamento =========
Acurácia: 0.9760765550239234
Matriz de confusão:
 [[0.99270073 0.00729927]
 [0.05555556 0.94444444]]
========== Estatísticas de validação ==========
Acurácia:  0.72
Matriz de confusão:
 [[0.70588235 0.29411765]
 [0.27272727 0.72727273]]
================================================
```

- Solver: *L-BFGS* e *LIBLINEAR* (resultado idêntico); Regularizador: *L2*.

```
================================================
Regulaziração: l2
========= Estatísticas de treinamento =========
Acurácia: 1.0
Matriz de confusão:
 [[1. 0.]
 [0. 1.]]
========== Estatísticas de validação ==========
Acurácia:  0.72
Matriz de confusão:
 [[0.76470588 0.23529412]
 [0.3030303  0.6969697 ]]
================================================
```

Percebe-se que os modelos que implementam termos de regularização foram mais precisos nos casos de validação, no entanto, o modelo com regularizador *L1* foi o único que não apresentou acurácia de 100% para os dados de treino, portanto estima-se que os outros modelos tenham apresentado *overfitting*.

3.2. Rede de Camada Rasa

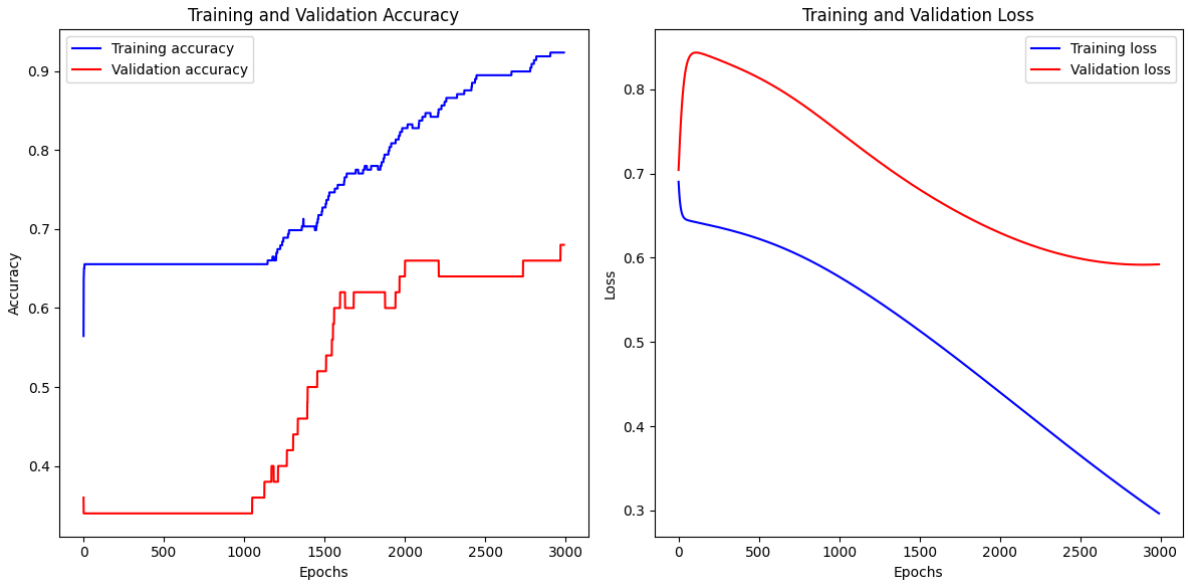
Para treinamento em cada época,o *batch* foi o total de imagens de treinamento. Também preferiu-se utilizar taxas de aprendizado relativamente pequenas (< 0.06) e épocas de treino maiores (> 800), com paradas antecipadas, porque em geral apresentaram resultados mais satisfatórios do que taxas de aprendizado maiores em menos épocas.

Ao todo, um modelo foi gerado para cada combinação de: 3 quantidades de neurônios na camada intermediária (60, 200 e 1000, 2000), 3 funções de ativação (sigmóide, tangente hiperbólica e ReLU), e 4 taxas de aprendizado (0.004, 0.008, 0.016, 0.06). Seus resultados foram filtrados e por fim de brevidade não serão incluídos no relatório.

3.2.1. Sigmóide

Em geral, o treinamento dos modelos que utilizaram ativação por sigmóide foi muito lento em relação aos outros, levando muitas épocas a mais para atingir um nível aceitável de valor da função de custo para os dados de validação.

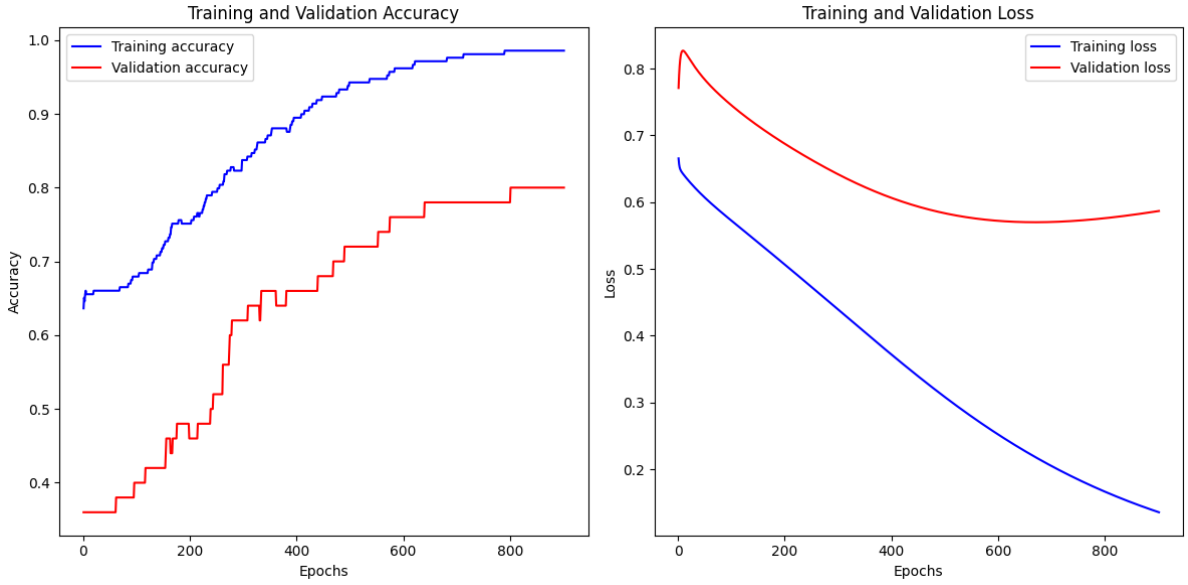
O melhor resultado foi obtido utilizando 60 neurônios e 0.008 de taxa de aprendizado em ~3000 épocas:



3.2.2. Tangente Hiperbólica

Em geral, os modelos obtidos utilizando ativação por tangente hiperbólica foram os mais eficientes, com métricas satisfatórias em modelos com poucos neurônios na camada intermediária, com treinamento em menos épocas, em comparação aos demais.

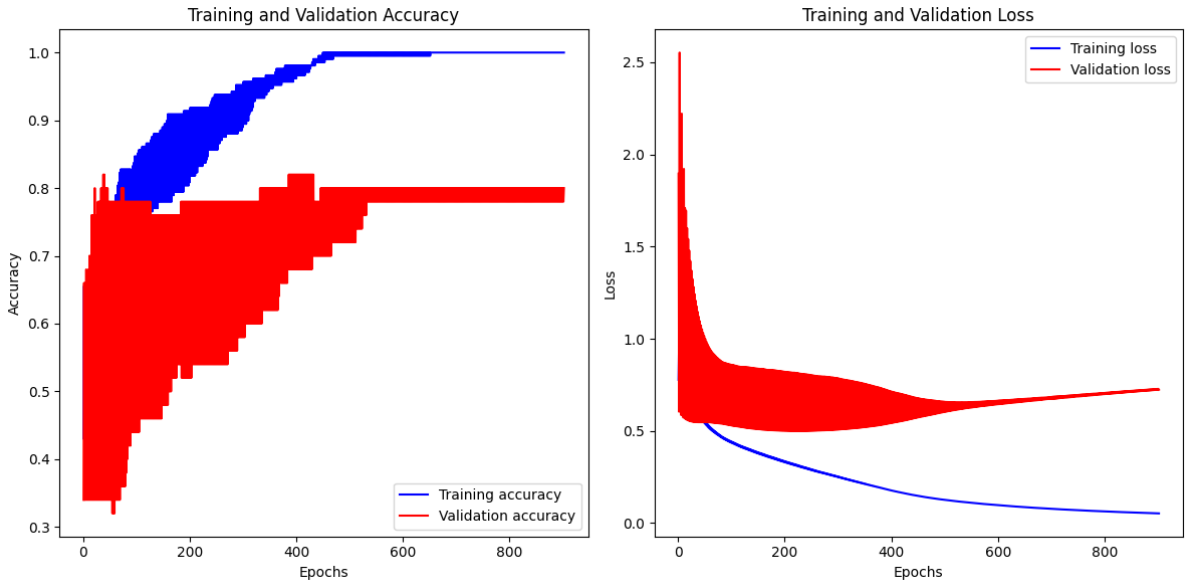
O melhor resultado foi obtido utilizando 200 neurônios e 0.008 de taxa de aprendizado em ~1000 épocas:



3.2.3. ReLU

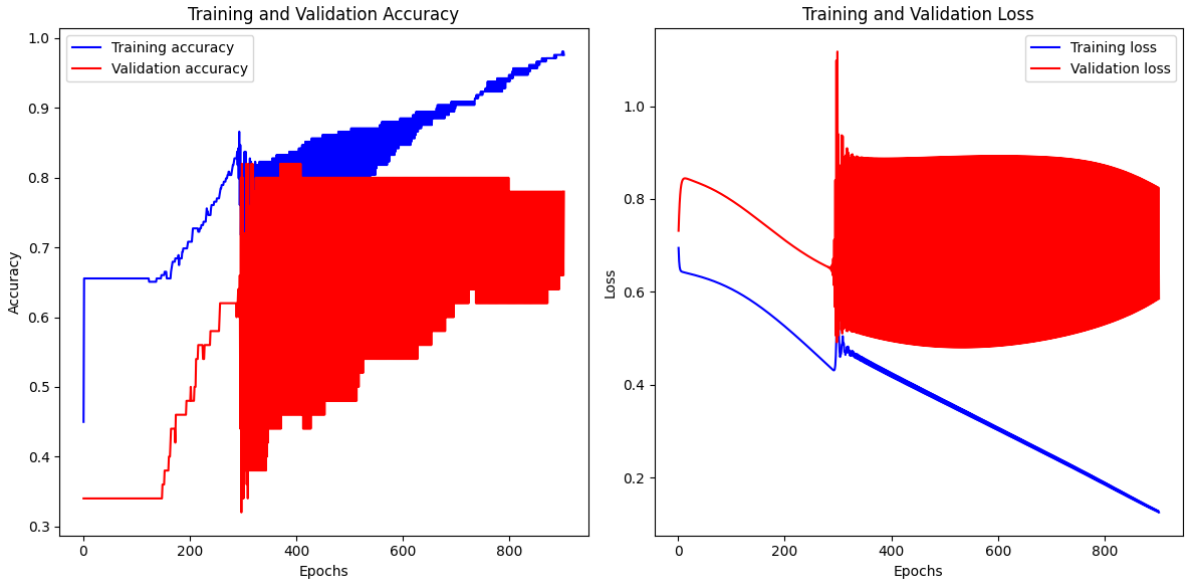
Em geral, os modelos que utilizaram ReLU também convergiram em bons resultados, porém seus gráficos de métricas foram os mais peculiares.

O melhor resultado foi obtido utilizando 1000 neurônios e 0.008 de taxa de aprendizado em ~1000 épocas, no entanto, a acurácia de treinamento convergiu em 100% e o valor da função de custo em quase 0, um indicativo de *overfitting*.



3.2.4. Exemplo de Modelo Filtrado

As seguintes métricas foram extraídas a partir de um modelo que utiliza ativação por sigmóide, que por conta de sua alta taxa de aprendizado (neste caso 0.016), apresenta este comportamento ruidoso, oscilatório.



3.3. Análise dos Resultados

Verifica-se, portanto, que o melhor modelo foi obtido utilizando uma rede neural de camada rasa, com ativação por tangente hiperbólica, de 200 neurônios na camada intermediária e 0.008 de taxa de aprendizado. Este modelo apresentou acurácia de quase 100% e 80% para os dados de treinamento e validação, respectivamente. E para os melhores casos de cada função de ativação, o comportamento do valor da função de custo foi muito similar.

Verifica-se também que, embora o modelo de regressão logística tenha tido uma acurácia de validação menor, é muito menos custoso e complexo do que um modelo de rede neural, apresentando um resultado de predição satisfatório, ainda mais quando levado em consideração que seu custo é muito inferior ao de uma rede neural e que a diferença para o melhor caso é de apenas 8% (72% contra 80%).

4. Instruções Para Execução dos Programas

4.1 Requisitos

É necessário ter instalado o *Python* com versão superior à 3.8, e *Pip Package Manager* com versão correspondente à versão instalada do *Python*. Então, rodar o seguinte comando instalará as dependências dos programas:

```
> pip3 install -r requirements.txt
```

4.2 Execução

Para rodar os programas, basta escolher um deles, *logistic_regression.py* ou *shallow_network.py*, e executá-los utilizando *Python*, por exemplo:

```
> python3 logistic_regression.py
```

Para fins de eficiência, a versão entregue do arquivo *shallow_network.py* só constrói e treina o modelo de rede neural mais “bem-sucedido”. Os trechos de código que faziam com que o modelo utilizasse outros parâmetros como taxa de treinamento e função de ativação foram comentados.