Compreensão de Linguagem Natural Americana

Pedro Escaleira
Departamento de Electrónica
Telecomunicações e Informática
Universidade de Aveiro
Aveiro, Portugal
escaleira@ua.pt

Rafael Simões
Departamento de Electrónica
Telecomunicações e Informática
Universidade de Aveiro
Aveiro, Portugal
rafaeljsimoes@ua.pt

***Abstract*—Aprendizagem Automática é uma área das ciências da computação com cada vez mais destaque nos dias que correm. Desde a sua utilização em campos como a saúde, sistemas bancários, *self driving cars*, nas aplicações de telemóvel do utilizador comum, entre outros, não é surpresa para ninguém que se tornou uma realidade sem a qual a sociedade em que vivemos seria muito diferente do que é hoje.**
Neste relatório, iremos descrever a utilização de alguns dos algoritmos mais conhecidos e usados nesta área e o seu comportamento perante diferentes situações, usando para isso um *dataset* simples de fotografias de gestos de linguagem gestual americana.

I. Introdução

Este trabalho, proposto pela professora Petia Georgieva, do Departamento de Electrónica Telecomunicações e Informática da Universidade de Aveiro, teve o intuito de consolidar e instigar a utilização e pesquisa dos conceitos apresentados na disciplina de Tópicos de Aprendizagem Automática. Desta forma, estudamos o comportamento dos algoritmos **Redes Neuronais**, ***Support Vector Machine*** e **Regressão Linear**, de forma a perceber a influência que alguns dos parâmetros destes teriam influencia na performance dos mesmos e como solucionar possíveis problemas que advém da utilização de cada um.

II. Ferramentas usadas

Este projecto foi executado usando-se *Python* como linguagem de referência. Contudo, para facilitar o nosso trabalho, usamos algumas bibliotecas *open source*:

- *Pandas*: usado para ler os dados dos *datasets* originalmente em *csv* para matrizes de *numpy*.
- *NumPy*: usado para fazer cálculos matriciais mais facilmente.
- *Matplotlib*: usado para criação facilitada de gráficos e figuras.
- *scikit-learn*: usado para estudar os algoritmos de aprendizagem automática descritos neste relatório.
- *Pickle*: usado para armazenar e carregar os modelos treinados em ficheiros binários.

III. Dados

As imagens usadas para fazer este estudo foram criadas, originalmente, por várias pessoas a reproduzirem múltiplas vezes os gestos referentes às letras pretendidas, com diferentes fundos, para uma maior diversidade de dados [1]. É possível observar um exemplo destas fotos, já com algum pós processamento, na figura 1.

A. Transformação do Dataset original

Pelo facto do *dataset* original conter um conjunto relativamente pequeno de dados (1704 imagens), este foi transformado num de maiores dimensões através de várias transformações feitas às imagens originais, sendo que o *dataset* obtido e com que trabalhamos possui um total de 34627 dados, distribuídos por 24 *labels*, numerados de 0 a 25 mapeados para as letras de A a Z alfabeticamente (foram excluídas as letras "J", correspondente ao *label* 9, e "Z", correspondente ao *label* 25, por necessitarem de movimentação gestual).

É de destacar também que os dados se encontram estruturados de forma similar aos do *MNIST*[2]: os objectos submetidos a classificação (neste caso, os gestos) estão centrados em imagens de 28x28 pixeis, em escala cinzenta, armazenados num ficheiro *csv*, onde a primeira coluna de cada linha corresponde ao *label* daquilo que está a ser classificado e as restantes colunas correspondem aos valores dos pixeis da

Fig. 1. Alguns exemplos de imagens, a cores, do *dataset*. Elas representam as letras do alfabeto americano em linguagem gestual que não necessitam de movimentação.

[1] https://github.com/mon95/Sign-Language-and-Static-gesture-recognition-using-sklearn

[2] http://yann.lecun.com/exdb/mnist/

imagem original, numa escala de 0 (preto) a 255 (branco), resultando num total de 785 colunas.

Este tratamento realizado sob os dados originais não foi feito por nós, estando já disponíveis no *Kaggle*[3].

B. Divisão dos dados para o estudo

Apesar dos dados obtidos no *Kaggle* se encontrarem divididos em dois *datasets*, um para treino e outro para testes, seguindo o método *Hold-out*, decidimos, tal como sugeridos nas aulas da disciplina, dividir os dados em três partes, de forma a obter um terceiro *dataset*, para *cross validation*, de forma a fazer a selecção do melhor modelo e usar os dados de teste apenas para teste do melhor modelo que encontrá-mos para cada algoritmo. Desta forma, a divisão que decidimos efectuar sobre os dados foi a recomendada na disciplina, isto é, aproximadamente 60% (20776 dados) de dados de treino e 20% de *cross validation* (6925 dados) e de teste (6925 dados), distribuídos de forma aleatória por cada um a partir dos dados iniciais. Como é possível constatar pela figura 2, os dados ficaram distribuídos de forma praticamente uniforme por cada um dos *labels* existentes em qualquer um dos *datasets*.

C. Forma como os dados foram explorados no estudo

Com os dados devidamente separados em três *datasets*, a forma como usamos cada um deles foi a seguinte:

- Os dados de treino foram usados para treinar os vários modelos que exploramos, mais concretamente.
- Os dados de *cross validation* foram usados para determinar o melhor modelo para cada variação de hiperparâmetros testada.
- Os dados de teste foram usados para perceber o quão boa a prestação modelo de cada algoritmo usado foi após fazer o *fine tuning* deste.

Para além disso, foi também usado o método de *Feature Scaling*, de forma a reduzir a escala das *features* (neste caso, os pixeis das imagens), dividindo cada uma delas por 255, já que é o valor máximo que um pixel pode possuir. Foi realizada esta normalização não com o intuito de trabalhar com todas as *features* na mesma escala, até porque já se encontravam, mas sim de forma a evitar *overflow* durante os cálculos realizados por cada algoritmo.

IV. Estudo com Redes Neuronais

Sendo que o pressuposto das Redes Neuronais é que tenham um bom desempenho, em termos da qualidade da prestação de modelos treinados usando o mesmo, foi a primeira abordagem que decidimos dar ao problema. O maior problema que esperávamos encontrar ao usar este algoritmo era, como seria de esperar, a sua pobre prestação temporal a fazer o treino de cada modelo testado.

[3]https://www.kaggle.com/datamunge/sign-language-mnist

A. Estrutura usada

Pelo facto de termos considerado que o problema não possuía uma complexidade extrema de ser resolvido, apresentando uma dificuldade similar ao do *MNIST*, onde a maior diferença se encontra no facto de neste caso haver uma maior quantidade de classes para identificar, decidimos não usar um modelo de *deep learning*, mas sim uma rede de apenas 3 *layers*:

- Uma *input layer* com 784 nós.
- Uma *hidden layer* com 50 nós (foi um dos parâmetros que variamos, mas nos outros estudos, foi sempre este o tamanho usado).
- Uma *ouput layer* com 24 nós (sendo que a letra "Z" era uma das duas que não podia ser representada estaticamente e correspondendo a um *label* numa das extremidades, não foi contabilizada na *ouput layer*).

Para além disso, a inicialização dos *weights* foi feita de forma aleatória em qualquer modelo e os valores dos hiperparâmetros usados por *default* foram:

- $\lambda = 0$
- Número de iterações: 1000
- *Activation Function*: *Logistic*
- *Batch Size*: 10

Os resultados dos **erros** apresentados em cada um dos próximos tópicos foram obtidos através da formula do ***Cross-Entropy Loss Function***, como é possível verificar na equação1.

B. Estudo da variação do alfa

A primeira abordagem que decidimos ter ao testar o uso deste algoritmo foi a de variar o valor de alfa, isto é, o valor da *learning rate*, já que este parâmetro dita o tamanho de cada *step* tomado em cada iteração do *gradient descent*, afectando muito directamente a prestação que o modelo treinado apresenta, isto é, tem um papel fundamental no quão boa ou má é feita a generalização do modelo treinado em relação aos dados de *input*. Desta forma, sendo que no inicio não tínhamos noção de quais os melhores intervalos de alfa para este modelo, decidimos fazer um primeiro treino para valores mais desfasados de alfa. Os erros obtidos nestes modelos podem ser observados no gráfico da figura 3, onde se pode perceber que o menor erro foi obtido para um valor de alfa $\alpha = 10^{-5}$, pelo que percebemos que os melhores valores de alfa seriam encontrados em torno desse valor.

Sendo assim, decidimos treinar vários modelos para valores de alfa abaixo e acima de $\alpha = 10^{-5}$, mais concretamente, para valores de alfa entre $5 * 10^{-6}$ e $1.5 * 10^{-5}$ com intervalos de 10^{-6} entre si. Os erros obtidos para cada um destes modelos podem ser consultados na figura 4.

O passo seguinte foi fazermos *retraining* do modelo que teve melhor prestação no passo anterior, ou seja, o que teve menor erro quando submetido aos dados de *cross validation*, neste caso para $\alpha = 1.4^{-5}$. Este passo foi essencial para diminuir ainda mais o erro e ter um modelo com uma ainda melhor prestação. Para isso, usamos o modelo já treinado, isto

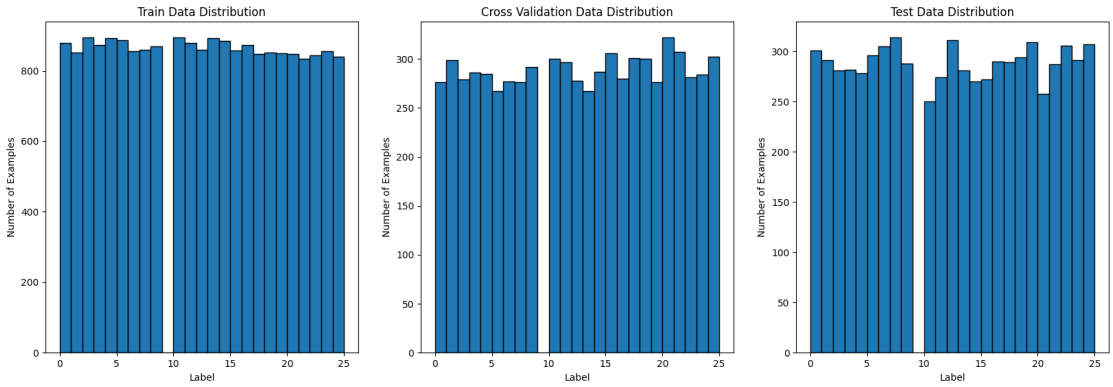

Fig. 2. Distribuição de dados por *label* por *dataset*.

$$E(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[-y_k^i log((h_\theta(x^i))_k) - (1-y_k^i)log(1-(h_\theta(x^i))_k)] \quad (1)$$

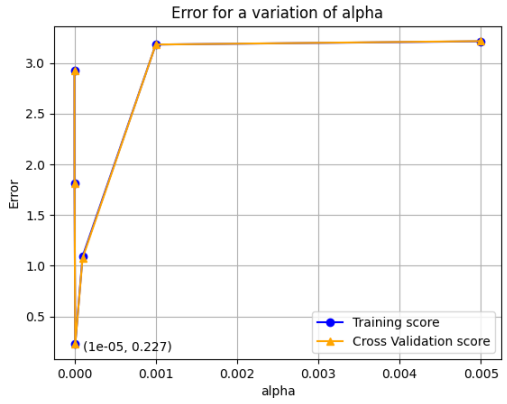

Fig. 3. Erro dos modelos obtidos para os valores de alfa $\alpha \in \{10^{-3}, 5*10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}\}$.

é, os *weights* obtidos no treino desse modelo e continuamos a treina-lo durante mais 10 000 iterações. Os valores da *cost function* ao longo das 11 000 iterações deste modelo podem ser consulados no gráfico da figura 5. Neste é impossível deixar de reparar que, no momento em que se iniciou o retreino do modelo e em algumas iterações após esse evento, houve um grande aumento do valor da *cost function*. Isto pode ser explicado pelo comportamento conhecido como *Catastrophic interference*, que é tido como o esquecimento total ou parcial daquilo que um modelo tinha aprendido num treino anterior a um novo treino com novos dados ou com novas classes [**?**]. Em termos de métricas, é possível verificar a performance do modelo perante os dados de teste, consultando os dados da tabela I. Duma forma geral, o modelo apresentou um valor aceitável em qualquer uma das métricas, sendo este valor igual para todas e igual a 97.7%.

Quanto aos valores da matriz de confusão deste modelo, com os dados de teste, podem ser consultados na tabela II. Nela, pode-se constatar que, de uma forma geral, o modelo praticamente adivinhou correctamente todas as classes.

C. Estudo da variação do tamanho da Hidden Layer

Decidimos também, como parte do nosso estudo, verificar o comportamento deste algoritmo para a variação do tamanho da *hidden layer*, e verificar como o maior ou menor tamanho desta afectava a prestação do nosso modelo, bem como a complexidade computacional do treino do mesmo. Para isso, usamos o valor do melhor alfa obtido no ponto anterior e variámos o valor do número de nós da *hidden layer* de 5 em 5, entre 5 e 70. Assim sendo, obtivemos o gráfico dos erros de treino e de validação apresentado da figura 6.

Dos resultados obtidos, é-nos possível entender que o valor do erro para os dados de validação começa a estagnar para

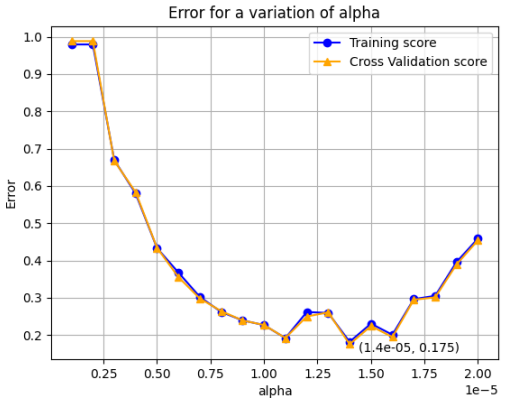

Fig. 4. Erro dos modelos obtidos para os valores de alfa $\alpha \in \{\alpha \in R \mid 5*10^{-6} \leq \alpha \leq 2*10^{-5} \mid \alpha * 10^6 \in N\}$.

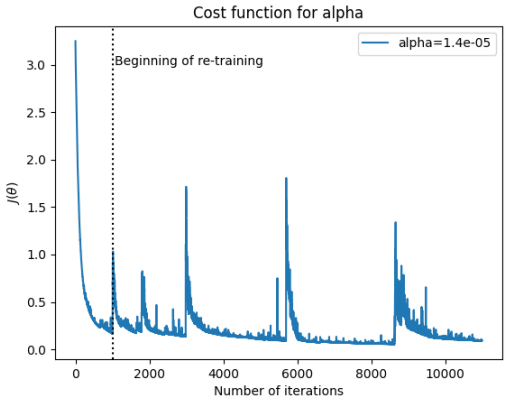

Fig. 5. *Cost function* para o alfa $\alpha = 1.4 * 10^{-5}$, medida em cada iteração do treino do modelo.

TABLE I
MÉTRICAS DE PERFORMANCE PARA $\alpha = 1.4 * 10^{-4}$

Class	Accuracy	Recall	Precision	F1 Score
0	1.0	1.0	0.997	0.998
1	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0
3	1.0	1.0	0.976	0.988
4	0.996	0.996	0.965	0.981
5	0.926	0.926	0.996	0.96
6	1.0	1.0	0.997	0.998
7	0.978	0.978	1.0	0.989
8	0.965	0.965	0.962	0.964
10	0.984	0.984	0.95	0.967
11	1.0	1.0	0.996	0.998
12	0.99	0.99	0.922	0.955
13	0.911	0.911	0.988	0.948
14	0.993	0.993	0.978	0.985
15	0.993	0.993	1.0	0.996
16	0.948	0.948	0.979	0.963
17	0.958	0.958	0.989	0.974
18	0.949	0.949	0.986	0.967
19	0.984	0.984	0.953	0.968
20	0.961	0.961	0.976	0.969
21	0.972	0.972	0.965	0.969
22	0.964	0.964	0.964	0.964
23	0.973	0.973	0.943	0.958
24	0.997	0.997	0.971	0.984
Macro Average	0.977	0.977	0.977	0.977

um número de nós acima de 40, inclusive. Uma possível explicação para este comportamento acontecer pode-se basear no facto de que, para um número de nós acima de 40, o custo do treino já praticamente convergiu. Como se pode perceber no gráfico da figura 7, para valores muito pequenos do número de nós, para 1000 iterações, a "rapidez" com que o custo converge é muito menor do que para valores mais elevados, sendo que para estes últimos, nas ultimas iterações, os custos são muito parecidos de modelo para modelo. Fizemos também um estudo da complexidade temporal associado ao tamanho da *hidden layer*, já que este está directamente relacionado com a complexidade computacional do treino, já que quantos mais nós houver nesta camada, maior será o número de cálculos matriciais feitos. Como demonstrado no gráfico da figura 8, conclui-se que o tempo de execução varia, duma forma geral, linearmente de acordo com o número de nós. Por conseguinte, num caso de estudo onde se privilegia o tempo com que o treino é executado em detrimento de alguma qualidade nas previsões feitas pelo modelo treinado, possivelmente seria aconselhável utilizar um número de nós mais reduzido, como por exemplo 40, obtendo-se um erro aceitável, mas claro, não tão baixo como seria se se usassem mais nós.

D. Estudo da variação do número de iterações

Ao obtermos o melhor valor de alfa, constatamos ao analisar o gráfico do custo de treino do melhor modelo ao longo das várias iterações que a *cost function* começava a convergia muito antes de completar 1000 iterações. Sendo assim, decidimos fazer um estudo da forma como o número de iterações iria afectar a performance do modelo. Na figura 9 é possível analisar o gráfico da evolução do erro de acordo com o número de iterações, para alfa $\alpha = 1.4 * 10^{-5}$. Neste, pode-se constatar que para mais de 500 iterações, o valor do erro começa a ser parecido entre as várias iterações. Sendo que o número de iterações está intimamente relacionado com a complexidade computacional do algoritmo, como se pode verificar no gráfico da figura 10, chegamos à conclusão de que é possível obter modelos com uma performance aceitável em menos tempo, se executados durante menos iterações.

De acordo com vários estudos feitos, o método de *early stopping* é já usado em muitos estudos e projectos que usam redes neuronais, dado o facto da complexidade computacional ser muito mais baixa e de automatizar o processo de seleccionar o melhor modelo de acordo com o número de iterações necessárias para a curva da *cost function* começa a convergir [**?**]. Desta forma, decidimos experimentar treinar um modelo nas mesmas condições, mas usando este método, de maneira a verificar se conseguia-mos obter resultados aceitáveis. Neste caso, usamos um *stopping criteria* inverso do terceiro descrito no estudo [**?**], isto é, o treino acaba se, passado um determinado número de iterações consecutivas, nc, o valor da função de custo não melhorar mais do que um determinado

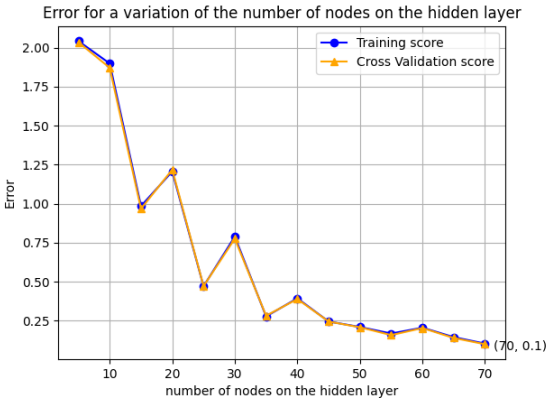

Fig. 6. Erro dos modelos obtidos para $n \in \{n \in N \mid 5 \leq n \leq 70 \mid n/5 \in N\}$, onde n representa o número de nós da *hidden layer*.

TABLE II
MATRIZ DE CONFUSÃO PARA $\alpha = 1.4 * 10^{-4}$

		Actual Class																						
	Class	0	1	2	3	4	5	6	7	8	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Predicted Class	0	301	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	291	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2	0	0	281	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	3	0	0	0	282	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	0	277	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	5	0	0	0	0	0	274	0	0	0	5	0	0	0	0	0	0	0	0	5	0	5	0	7	0
	6	0	0	0	0	0	0	305	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	1	307	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8	1	0	0	0	0	0	0	0	278	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0
	10	0	0	0	0	0	0	0	0	4	246	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	11	0	0	0	0	0	0	0	0	0	0	274	0	0	0	0	0	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0	0	0	0	308	0	0	0	1	0	2	0	0	0	0	0	0
	13	0	0	0	0	0	0	0	0	0	0	0	22	256	0	0	3	0	0	0	0	0	0	0	0
	14	0	0	0	0	0	0	0	0	0	0	0	0	2	268	0	0	0	0	0	0	0	0	0	0
	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	270	1	1	0	0	0	0	0	0	0
	16	0	0	0	5	0	0	0	0	0	0	0	4	0	6	0	275	0	0	0	0	0	0	0	0
	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	277	0	0	6	0	6	0	0
	18	0	0	0	0	10	0	0	0	1	0	0	0	1	0	0	1	0	279	0	0	0	0	0	2
	19	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	304	0	0	0	4	0
	20	0	0	0	1	0	0	0	0	0	8	0	0	0	0	0	0	1	0	0	248	0	0	0	0
	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	279	5	0	3
	22	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	295	6	4
	23	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	5	0	283	0
	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	306

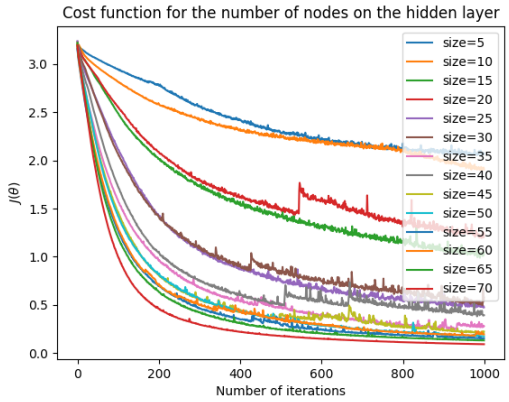

Fig. 7. *Cost function* para $n \in \{n \in N \mid 5 \leq n \leq 70 \mid n/5 \in N\}$, onde n representa o número de nós da *hidden layer*, medida em cada iteração do treino de cada modelo.

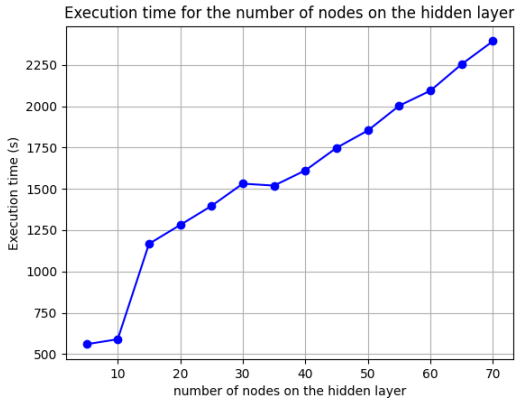

Fig. 8. Tempo de execução para $n \in \{n \in N \mid 5 \leq n \leq 70 \mid n/5 \in N\}$, onde n representa o número de nós da *hidden layer*.

valor predefinido, t (tolerância). Para $nc = 30$, $t = 10^{-4}$ e $\alpha = 1.4 * 10^{-5}$, o modelo treinou durante 778 iterações e a performance obtida perante os dados de teste pode ser consultada na tabela III. Como seria de esperar, a performance não foi tão boa como para o modelo que usou 1000 iterações para o mesmo valor de alfa, mas pode ser aceitável em certos casos de uso, ainda para mais tendo em conta que o tempo de execução é muito mais reduzido do que fazer a totalidade de 1000 iterações.

E. Estudo da variação do tamanho da batch

Por ultimo, decidimos analisar como o tamanho da *batch* poderia influenciar a qualidade do modelo obtido. Para isso, usamos o melhor alfa já anteriormente obtido e treinamos os modelos durante 1000 iterações. De acordo com estudos já feitos nesta área, é possível aumentar o tamanho da *batch* durante o processo de treino ao invés da diminuição da *learning rate* e, mesmo assim, obter resultados similares, mas num menor intervalo de tempo, já que um maior tamanho da *batch* implica menos actualizações dos parâmetros de treino [**?**]. Contudo, nós neste caso, e por uma questão de facilidade e limitações da ferramenta que usamos, o *scikit-learn*, fizemos um estudo para o treino de vários modelos com tamanhos de *batch* distintos entre si e constantes durante o treino, para um mesmo valor de alfa. O valor dos erros obtidos podem ser consultados no gráfico da figura 11. Numa primeira análise

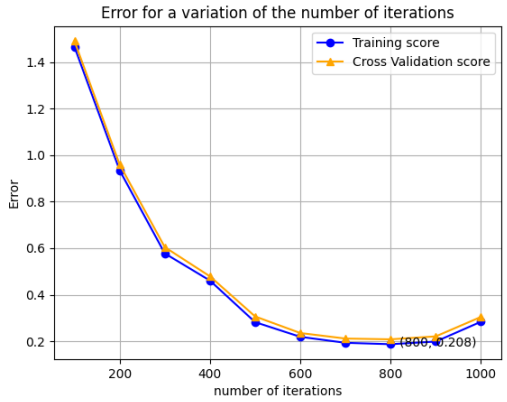

Fig. 9. Erro dos modelos obtidos para $i \in \{i \in N \mid 100 \leq i \leq 1000 \mid i/100 \in N\}$, onde i representa o número de iterações.

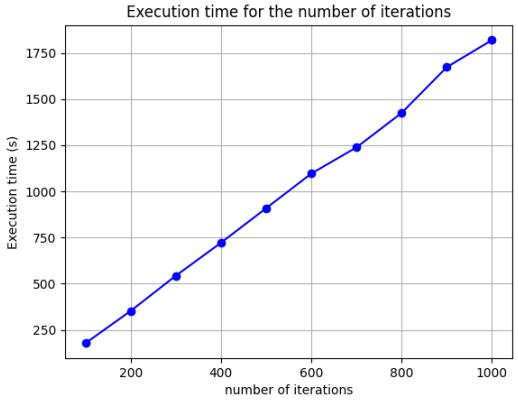

Fig. 10. Tempo de execução para $i \in \{i \in N \mid 100 \leq i \leq 1000 \mid i/100 \in N\}$, onde i representa o número de iterações.

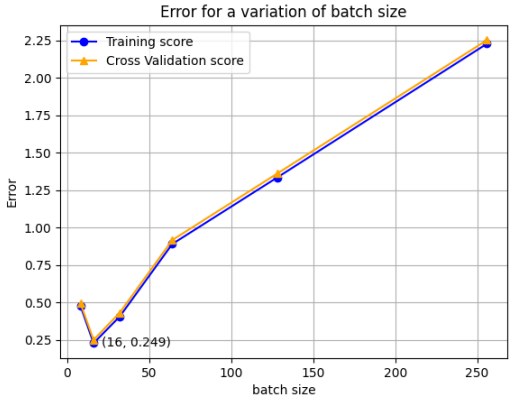

Fig. 11. Erro dos modelos obtidos para $b \in \{b \in N \mid 8 \leq b \leq 256 \mid \sqrt{b} \in N\}$, onde b representa o tamanho da *batch* usada.

destes resultados, acha-mos estranho, já que esperávamos que o erro baixasse e não subisse, pelo menos para os dados de treino, ao aumentarmos o tamanho da *batch*. Contudo, ao

TABLE III
MÉTRICAS DE PERFORMANCE PARA UM MODELO CUJO TREINO USOU *early stopping* COM $nc = 30$, $t = 10^{-4}$ E $\alpha = 1.4 * 10^{-5}$ E FOI EXECUTADO DURANTE 778 ITERAÇÕES

Class	Accuracy	Recall	Precision	F1 Score
0	1.0	1.0	0.974	0.987
1	0.911	0.911	1.0	0.953
2	0.989	0.989	0.952	0.97
3	1.0	1.0	0.959	0.979
4	0.968	0.968	0.957	0.962
5	0.943	0.943	0.969	0.955
6	0.918	0.918	0.982	0.949
7	0.917	0.917	0.886	0.901
8	0.979	0.979	0.979	0.979
10	0.912	0.912	0.954	0.933
11	0.978	0.978	0.931	0.954
12	0.707	0.707	0.952	0.812
13	0.701	0.701	0.99	0.821
14	0.878	0.878	0.967	0.92
15	0.926	0.926	0.984	0.955
16	0.997	0.997	0.963	0.98
17	0.779	0.779	0.996	0.874
18	0.959	0.959	0.597	0.736
19	0.968	0.968	0.824	0.89
20	0.938	0.938	0.864	0.9
21	0.927	0.927	0.818	0.869
22	0.886	0.886	0.858	0.871
23	0.876	0.876	0.985	0.927
24	0.935	0.935	0.96	0.947
Macro Average	0.916	0.916	0.929	0.918

analisarmos o gráfico da função de custo, verificamos que para um tamanho de *batch* mais elevado, a função convergia mais lentamente, como se pode verificar no gráfico da figura 12. Sendo assim, teríamos duas hipóteses: aumentar o número de iterações, o que não era pretendido neste caso, porque queríamos verificar se era possível obter um menor tempo de treino do que com *batchs* mais pequenas, ou aumentar o valor da *learning rate*, para um valor fixo do tamanho da *batch*, de forma a que a função de custo convergi-se mais rapidamente, tendo sido este o procedimento tomado e cujos resultados podem ser consultados no gráfico de erros da figura 13 e na tabela de métricas do modelo com menor erro (no *test dataset*) IV.

Apesar de não se ter obtido uma tão boa performance como quando utilizado um tamanho de *batch* mais pequeno (97.7% de *Accuracy* vs. 93.5% neste caso), é de notar que a performance é elevada e pode ser aceitável em muitos estudos, onde seja dada mais importância à rapidez com que o modelo é treinado. Como se pode verificar no gráfico da figura 14, a rapidez com que o treino é feito diminui substancialmente com o aumento do *batch size*.

F. Conclusões

Um ponto importante a destacar é que não fizemos qualquer estudo da variação do lambda [4]. Isto deve-se ao facto de não ter havido uma discrepância muito notável entre os erros de treino e de *cross validation*, ou seja, não obtivemos qualquer tipo de *overfit*. Este problema poderá advir da *data augmentation* que

[4]parâmetro de regularização

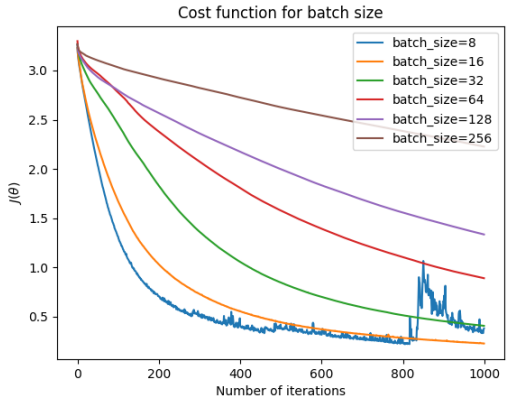

Fig. 12. *Cost function* para $b \in \{b \in N \mid 8 \leq b \leq 256 \mid \sqrt{b} \in N\}$, onde b representa o tamanho da *batch* usada.

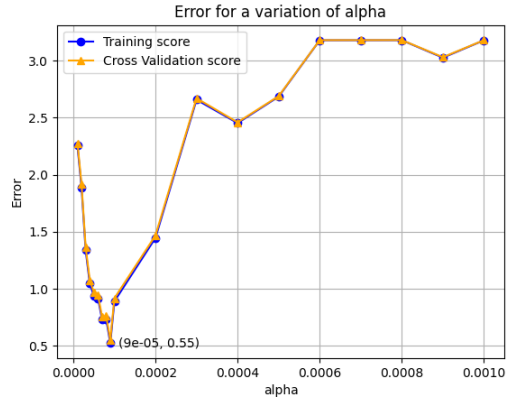

Fig. 13. Erro dos modelos obtidos para os valores de alfa $\alpha \in \{\alpha \in R \mid 10^{-4} \leq \alpha \leq 10^{-3} \mid \alpha * 10^4 \in N\} \cup \{\alpha \in R \mid 9 * 10^{-5} \leq \alpha \leq 10^{-5} \mid \alpha * 10^5 \in N\}$ e para *batch size* $b = 256$.

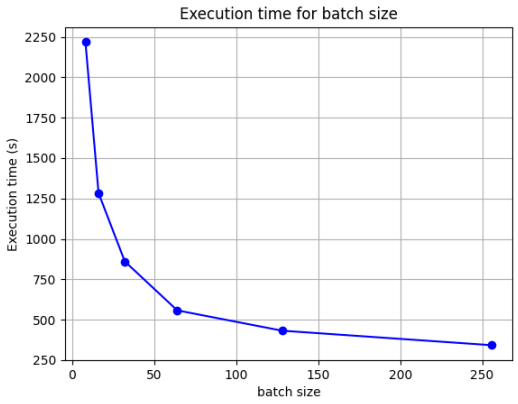

Fig. 14. Tempo de execução para $b \in \{b \in N \mid 8 \leq b \leq 256 \mid \sqrt{b} \in N\}$, onde b representa o tamanho da *batch* usada.

foi feita aos dados originais, sendo que, ao os novos dados estarem distribuídos aleatoriamente pelos três *datasets*, tenha provocado que os modelos treinados tiveram contacto com

TABLE IV
MÉTRICAS DE PERFORMANCE PARA *batch size* $b = 256$ E ALFA $\alpha = 9 * 10^{-5}$

Class	Accuracy	Recall	Precision	F1 Score
0	1.0	1.0	0.98	0.99
1	0.979	0.979	0.966	0.973
2	0.996	0.996	1.0	0.998
3	1.0	1.0	0.986	0.993
4	0.917	0.917	0.992	0.953
5	0.932	0.932	0.962	0.947
6	0.954	0.954	0.942	0.948
7	0.949	0.949	0.99	0.969
8	0.962	0.962	0.982	0.972
10	0.948	0.948	0.868	0.906
11	0.945	0.945	0.827	0.882
12	0.932	0.932	0.924	0.928
13	0.936	0.936	0.916	0.926
14	0.981	0.981	0.996	0.989
15	0.908	0.908	0.888	0.898
16	0.938	0.938	0.925	0.932
17	0.893	0.893	0.819	0.854
18	0.874	0.874	0.918	0.895
19	0.919	0.919	0.937	0.928
20	0.919	0.919	0.919	0.919
21	0.857	0.857	0.908	0.882
22	0.83	0.83	0.951	0.887
23	0.893	0.893	0.935	0.914
24	0.98	0.98	0.929	0.954
Macro Average	0.935	0.935	0.936	0.935

todas as imagens originais, não existindo qualquer novidade quando foi submetido aos dados de validação e de teste. Outro problema a apontar ao uso deste algoritmo é o facto da computação ser muito pesada, ainda para mais se feita com recurso a um *CPU* ao invés dum *GPU*. A principal diferença entre usar um *CPU* e um *GPU* está no facto deste ultimo usar uma muito menor quantidade de núcleos que o primeiro. Sendo que as computações mais exigente das Redes Neuronais são cálculos matriciais, que podem ser paralelizados, um *GPU* tem a capacidade de paralelizar uma muito maior quantidade de cálculos em simultâneo que um *CPU* pelo simples facto de possuir muitos mais núcleos para os fazer. Existem também outras razões para esses cálculos serem executados mais rapidamente em *GPUs*, como a maior largura de banda e a maior dimensão e rapidez que os seus registos de memória possuem [**?**]. Sendo que usamos o *scikit-learn* para fazer o estudo e este apenas executa as computações em *CPU*, foi extremamente penoso proceder ao treino dos vários modelos apresentados. Chegamos por isso à conclusão de que, para este algoritmo, a resposta seja a utilização duma biblioteca como o *TensorFlow*, que permite a utilização do *GPU* do computador para treinar os modelos de Redes Neuronais. Em termos dos valores dos parâmetros a usar neste algoritmo para este problema, temos duas visões, que se baseiam no *trade off* que tem de ser feito entre a complexidade temporal e a qualidade dos resultados obtidos:

- Por um lado, se quisermos utilizar um modelo para um projecto onde seja privilegiada a qualidade dos resultados obtidos em detrimento do custo temporal que o treino do modelo necessita, então é recomendada a utilização dum

valor de alfa reduzido, um *batch size* mais pequeno, um número de iterações elevado, sem *early stop* e uma *hidden layer* com um tamanho razoável (não muito elevado, para não ocorrer *overfit*).

- Por outro lado, se pretendemos usar o modelo num projecto onde seja privilegiado custo temporal e onde é aceitável obter resultados menos satisfatórios, chegamos à conclusão de que a utilização dum alfa com um valor mais elevado, com um tamanho de *batch* grande, com um número de iterações determinadas por uma regra de *early stop* e uma *hidden layer* de tamanho médio, levam à obtenção dum modelo que não possui a melhor performance possível, mas razoável e cujo tempo de treino é aceitável.

Claro que estas conclusões, apesar de para este problema em específico, podem ser generalizadas para outros problemas do mesmo género.

REFERENCES

[1] Z. Chen, B. Liu, R. Brachman, P. Stone, and F. Rossi, *Lifelong Machine Learning: Second Edition*. Morgan Claypool, 2018.

[2] L. Prechelt, "Early stopping - but when?" 03 2000.

[3] S. Smith, P. jan Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," 2018. [Online]. Available: https://openreview.net/pdf?id=B1Yy1BxCZ

[4] A. Kayid, Y. Khaled, and M. Elmahdy, "Performance of cpus/gpus for deep learning workloads," 05 2018.