TP3 - Redes Neurais Artificiais para Classificação

Ramon Gonçalves Gonze

3 de dezembro de 2017

Resumo

O objetivo deste trabalho é apresentar uma rede neural artificial que resolva um problema de classificação. A maior parte do trabalho se deu em parametrizar a rede, e escolher de forma adequada quais valores conseguem classificar da melhor forma possível um exemplo fornecido para a rede. Foi utilizada uma rede MLP (Multi Layer Perceptron) com 3 camadas. Os resultados dos experimentos são apresentados ao final, juntamente com a análise sobre a variação dos parâmetros como funções de ativação dos neurônios, quantidade de épocas, números de neurônios nas camadas, entre outros, e seus respectivos impactos.

1 Introdução

Os problemas de classificação são aqueles em que, dado um conjunto de atributos de uma determinada entidade, deve-se identificar a qual classe ela pertenca. Na computação, o reconhecimento de padrões nos atributos dessas entidades é um dos focos da utilização de redes neurais artificiais.

Como citado acima, a escolha da MLP se deu devido ao fato de ser possível resolver problemas não-lineares, por possuir funções de ativação dos neurônios não-lineares. Isto não seria possível por exemplo, em uma rede de uma camada, que resolve somente problemas lineares.

Os parâmetros escolhidos serão descritos nas sessões **2** e **4**, assim como as relações entre cada um. A **Seção 4** também contém os experimentos e a análise dos resultados. Por fim, a última seção conclui todo o trabalho realizado, exibindo as dificuldades encontadas e os resultados gerais sobre a utilização de redes neurais artificiais para a resolução de um problema de classificação

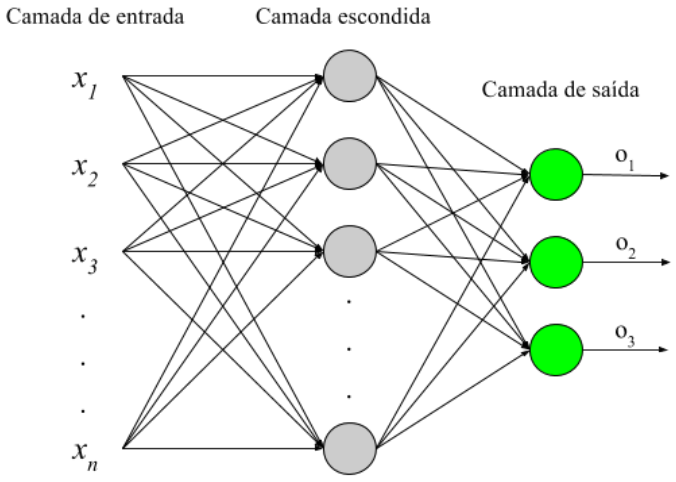

Figura 1: Exemplo de uma rede MLP de 3 camadas, uma rede *feedfoward*. Os valores $x_1, x_2, \ldots, x_n$ são as entradas, e os valores de o_1, o_2 e o_3 são as saídas da rede.

2 Modelagem

Nesta seção são descritas as técnicas e abordagens utilizadas para compor a arquitetura da rede.

2.1 Neurônios

Como já dito antes, foram utilizadas 3 camadas, uma de entrada, uma escondida e a camada de saída. Os neurônios da camada de entrada possuem uma função de ativação linear, ou seja, $f(x) = x$. Já os neurônios da camada escondida, possuem uma sigmóide, escolhida para introduzir a não-linearidade na rede. Por fim, para a última camada, foi utilizado o *softmax*, onde o valor de saída de um neurônio é normalizado pela soma das saídas de todos os neurônios.

Em relação ao número de neurônios, foram testadas diversas combinações para as camadas de entrada e escondida. Como o problema é de classificação, e o dataset testado nesse trabalho possui 7 classes, foram inseridos 7 neurônios na camada de saída. A **Seção 4.1** demonstra as melhores quantias de neurônios para as duas primeiras camadas.

2.2 Atualização dos pesos

Com o intuito de conseguir o menor erro possível, ou seja, maximizar o número de acerto ao predizer a classificação de um exemplo, foi utilizada uma abordagem de *mini-batches*. A atualização dos pesos da rede são feitas após x quantidade de exemplos vistos pela rede. A **Seção 4.4** demonstrará a utilização de vários tamanhos de mini-batches, e qual valor alcançou o melhor resultado.

2.3 Validação cruzada

A validação cruzada é utilizada em algoritmos de aprendizagem para se ter uma confiança maior nos resultados. Ao fazer o treinamento com um determinado conjunto de exemplos e também avaliar com ele, não conseguimos verificar a real generalização da rede, pois a rede pode "decorar"os exemplos do treinamento, fazendo com que a avaliação se torne redundante. Portanto, é necessário verificar a capacidade de generalização da rede em um conjunto de exemplos disjunto do primeiro.

A separação do dataset foi feita da seguinte forma:

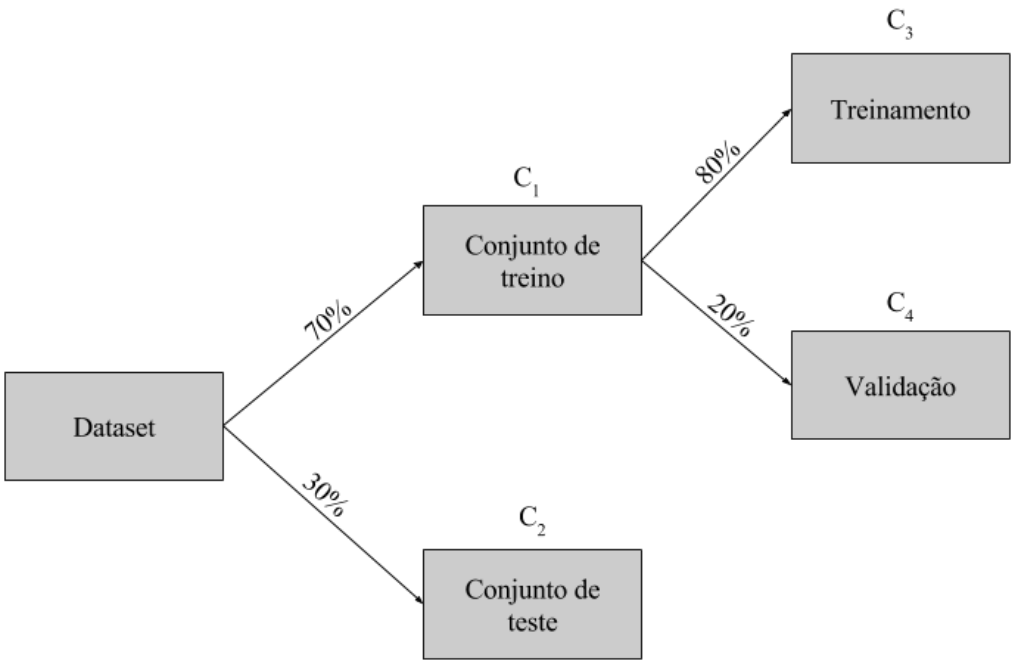

Figura 2: Divisão do dataset em conjuntos para o treinamento e a validação da rede. As porcentagens nas setas indicam a quantidade de exemplos que foi passada para cada conjunto.

Seja C_1, C_2, C_3 e C_4 os cojuntos da Figura 2. O treinamento da rede é feito sob o conjunto C_3, e após x exemplos passados na rede (tamanho do mini-batch), a acurária é avaliada no conjunto C_4. Essa avaliação com o C_4 faz com que a generalização se torne mais eficaz ainda, pelo motivo explicado no primeiro parágrafo desta seção. Ao final de todas as épocas, é testada a capacidade de generalização da rede no conjunto C_2.

Além da separação do dataset, em cada época, os exemplos do conjunto C_3 são passados para a rede em ordens diferentes.

3 Base de Dados e Execução

O problema de classificação da base de dados utilizada consiste em identificar em que parte da célula determinada proteína pode ser encontrada. A base possuí 1429 exemplos (proteínas), cada uma descrita por 8 atributos. Há 7 partes (considere **classes**) possíveis onde uma proteína pode ser encontrada, conforme a tabela abaixo:

Classe	Descrição
CYT	Citoplasma
MIT	Mitocôndria
ME1 ME2 ME3	Membrana da proteína
EXC	Fora da célula
NUC	Núcleo

Tabela 1: Descrição das partes onde uma proteína pode ser encontrada na célula.

O trabalho foi implementado em Python na versão 3.5.2. Para executar o programa, é necessário ter instalada as bibliotecas Keras (versão 2.1.2 ou superior) e Tensorflow (versão 1.4.0 ou superior). O programa possui somente 1 arquivo, denominado **tp3.py**. Ela recebe como parâmetro o nome do arquivo de entrada. O programa deve ser executado segundo o escopo:

```
python3 tp3.py <diretório/base_de_dados>
```

Exemplo de execução do programa:

```
python3 tp3.py yeast_modified.csv
```

A saída do algoritmo é feita pela saída padrão (**stdout**). A cada época, são imprimidos os valores de perda (função de custo) e acurácia no conjunto de teste, perda e acurácia no conjunto de validação. Ao final de todas as épocas, é feita a validação final no conjunto de testes, para medir o nível de generalização da rede.

4 Experimentos

Os experimentos foram realizados em um ambiente Linux utilizando a distribuição Ubuntu 16.04 LTS, em um computador com processador Intel Core i3, de 2.1GHz e 6GB de RAM.

4.1 Experimento 1 - Número de neurônios

Como já dito na **Seção 2.1**, a camada de saída possui 7 neurônios, e este valor não foi variado. O gráfico abaixo demonstra o resultado da variação dos neurônios na camadas escondida:

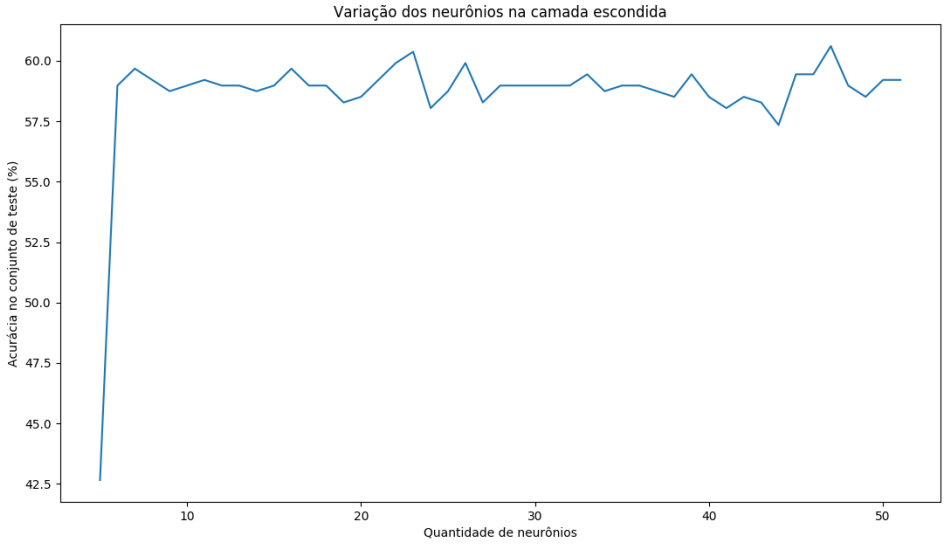

Figura 3: Relação entre quantidade de neurônios na camada escondida vs. capacidade de generalização da rede. Foram fixados 10 neurônios na camada de entrada, mini-batch de tamanho 10 e 200 épocas.

Podemos observar a partir do gráfico da Figura 3 que a partir de 8 neurônios na camada escondida, a variação da acurácia da rede não é tão distante de 5%. Quanto maior o número de neurônios, maior o tempo de execução. Portanto, como não há muito ganho com o aumento de neurônios, optou-se por utilizar **10** neurônios na camada escondida.

4.2 Experimento 2 - Número de camadas escondidas

A decisão de utilizar uma rede com somente uma camada escondida se deu pelo resultado do gráfico da Figura 4 abaixo:

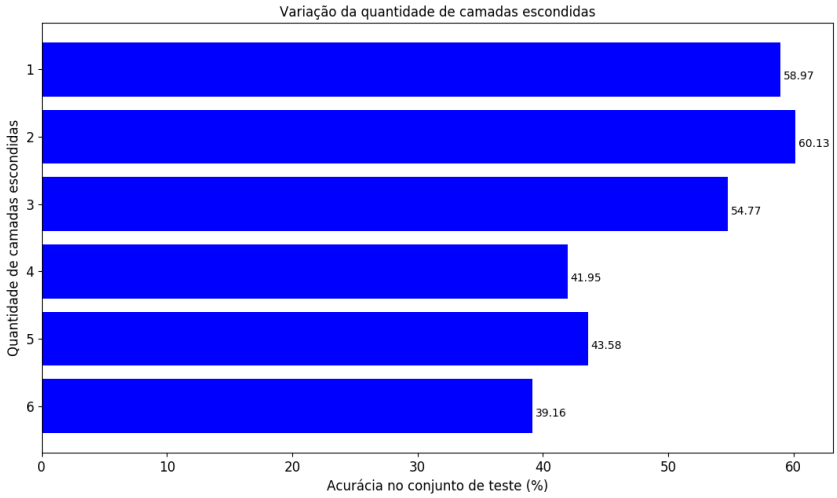

Figura 4: Relação da variação da quantidade de camadas escondidas vs. capacidade de generalização da rede. Todas as camadas escondidas possuíam 10 neurônios, sigmóide como função de ativação, e foi utilizado um mini-batch de tamanho 10 e 200 épocas.

É possível observar que simplesmente um aumento no número de camadas escondidas não

leva a um melhor desempenho, pelo contrário, o resultado se mostrou pior. Um dos fatores que influênciam essa piora na acurácia, é o número de épocas que foi fixado. Quanto maior o número de camadas, mais épocas são necessárias para atualizar todos os pesos com eficiência. Porém, quanto mais camadas e épocas, mais caro é o custo computacional. Optou-se por utilizar somente **1** camada escondida.

4.3 Experimento 3 - Taxa de aprendizagem

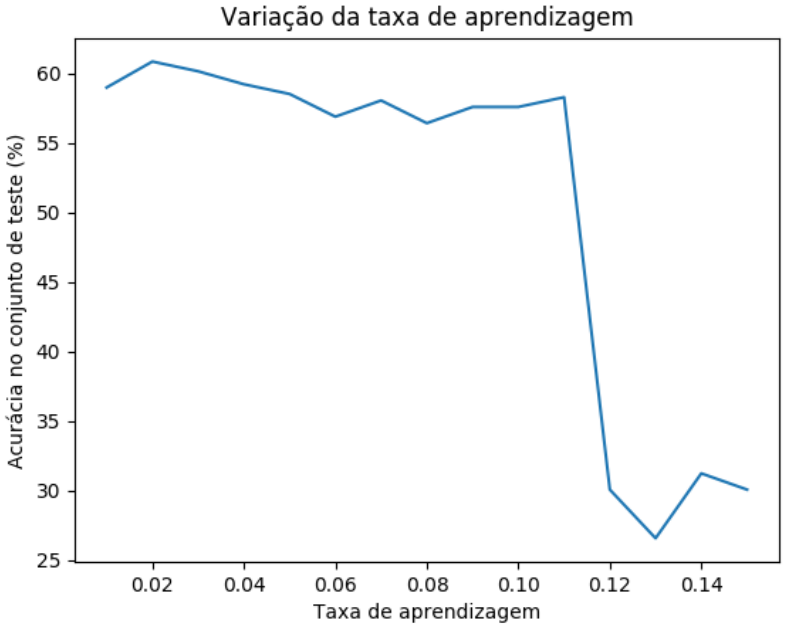

Figura 5: Variação da taxa de aprendizagem entre 1-15%. Os outros parâmetros utilizados foram os melhores obtidos nos experimentos anteriores, mini-batch de tamanho 10 e 200 épocas.

A partir do gráfico da Figura 5 acima, pode-se observar que a taxa de aprendizagem de 1% até 10% houve uma leve queda consecutiva na acurácia. Porém, a partir de 11, 12%, a queda em relação às menores taxas foi grande. Com uma taxa de aprendizagem muito alta e poucas épocas, a rede não tem tempo suficiente para alterar os pesos de forma adequada, o que resulta em uma generalização ruim. Optou-se por utilizar uma taxa de **1%** (0.01).

4.4 Experimento 4 - *Stochastic gradient descent* vs. mini-batches

Este experimento teve o intuito de averiguar qual o custo-benefício entre melhora da solução e custo computacional da abordagem do *Stochastic gradient descent* (SGD).

Método	Acurácia	Tempo de execução (min)
SGD	58.74 %	12.6
mini-batch = 10	59.90 %	1.34
mini-batch = 50	58.97 %	0.10
mini-batch = 100	58.65 %	0.6
mini-batch = 300	55.47 %	0.4

Tabela 2: Relação custo-benefício da utilização do SGD e mini-batches. Os outros parâmetros são os melhores obtidos nos experimentos anteriores.

Os resultados da Tabela 2 acima foram totalmente ao contrário do esperado. Esperava-se que com o SGD a acurácia obtida fosse a mais alta, e com um mini-batch de tamanho maior, um acurácia baixa. Realmente, quanto o mini-batch é extremamente grande (como no caso de

300), a acurácia começa a cair. Porém, entre o SGD e o mini-batch = 50, não houve alteração significativa. Esses resultados podem ser explicado talvez pela pouca quantidade de épocas (200). Por fim, optou-se por utilizar um mini-batch de tamanho **10**.

4.5 Experimento 5 - Overfiting

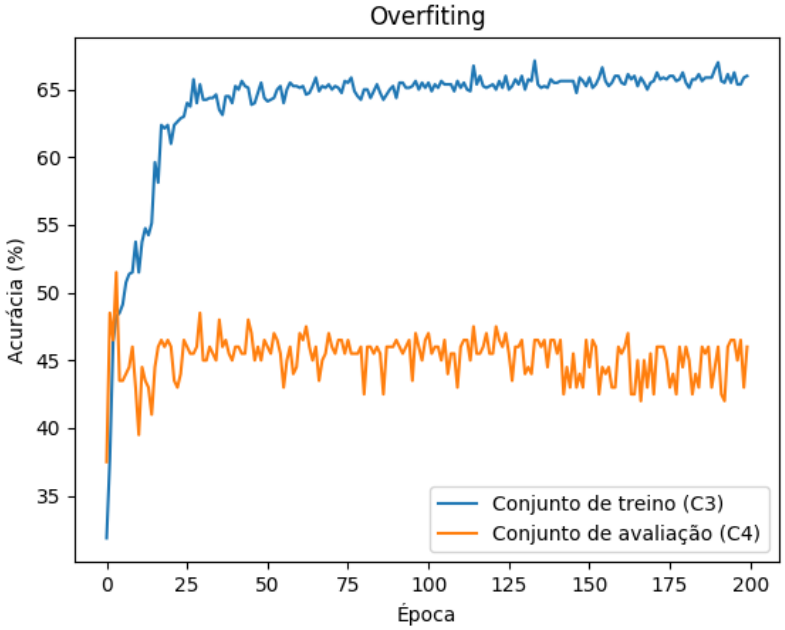

Figura 6: Overfiting existente no desempenho rede. Para este gráfico, o resultado obtido no conjunto de teste (C_2), foi de **58.28%**. Considere C_2, C_3 e C_4 os mesmos conjuntos definidos na Figura 2.

Pode-se observar no gráfico da Figura 6 acima que está havendo overfiting no desempenho da rede. A variação da acurácia nos conjuntos de treinamento e teste (em torno de 20%), demonstram que a rede não está conseguindo generalizar de forma suficiente os dados da base de dados. Um dos motivos seria o desbalanceamento da base de dados, ou seja, há muitos exemplos para algumas classes e poucos exemplos para outras.

5 Conclusão

A partir dos experimentos realizados e as análises sobre as mudanças de parâmetros, pode-se concluir que a rede neural artifical desenvolvida se aproximou da acurácia média obtida pela comunidade da computação na base de dados utilizada (em torno de 60%). Isso significa que, dependendo do objetivo a ser alcançado, é mais efetivo se ter um alto poder de processamento ao invés de gastar tempo tentando desenvolver otimizações na rede.

Houveram dificuldades no começo da implementação, pelo fato de a ferramenta que deu suporte para a criação da rede (Keras) nunca ter sido utilizada na prática antes. Mas ao longo da aprendizagem sobre a utilização desta biblioteca, o trabalho se tornou basicamente testar parâmetros diferentes.

Este trabalho apresentou somente a *surface* sobre aprendizado em computação, e ainda há muito material para se aprofundar no conhecimento sobre *deep* learning.

Referências

[1] HUANG D. S; LI K; IRWIN W. G. **Intelligent Computing in Signal Processing and**

Pattern Recognition: International Conference on Intelligence Computing, ICIC 2006 Kunming, China, August, 2006. Kunming: Springer-Verlag, 2006.

[2] **Keras: The Python Deep Learning library**. Disponível em: <https://keras.io/>. Acesso em: 3 de dez. 2017.