Universidade de São Paulo (USP) – Pós-Graduação ICMC, São Carlos
SCC5809 – Redes Neurais

Profa. Dra. Roseli Aparecida Francelin Romero
Discente: Damares C. Oliveira de Resende
Número USP: 11022990

Projeto I – Relatório das Atividades

O objetivo desse projeto é codificar uma rede neural multi-camadas (MLP) e testar sua performance em duas bases de dados conhecidas na área. São estas a base *wine*, a qual possui 13 atributos, 178 instâncias e 3 classes e é usada para classificação; e a base *default_features_1059_tracks*, a qual possui 66 atributos, 1059 instâncias e 2 rótulos indicando latitude e longitude que são estimados por regressão.

Pré-processamento:

Antes de passar as bases para a rede neural elas devem ser pré-processadas. No caso da base *wine*, os rótulos das classes foram transformados para outro domínio, onde a classe 1 para a rede é identificada como [0 0 1], a classe 2 como [0 1 0] e a classe 3 como [1 0 0]. Isso é importante pois esses vetores são linearmente independentes e facilitam assim o aprendizado da rede.

Outro método utilizado foi a normalização. Isso é crucial para o aprendizado nessa base de dados pois a variância entre os domínios dos atributos é muito alta, e caso a normalização não seja feita, a derivada da sigmoide tenderá a zero, o que implica em uma variação irrelevante nos pesos fazendo com que eles estagnem. A normalização também foi aplicada à base de dados *default_features_1059_tracks*. Em ambos casos normalizou-se a base subtraindo de cada valor o valor mínimo e dividindo o resultado pela diferença entre o máximo e mínimo.

Codificação:

O algoritmo foi implementado em Python sem o auxílio de bibliotecas prontas para a realização do aprendizado. Contudo, usou-se a biblioteca *pandas* para ler as bases de um arquivo *.csv* e do *sklearn.model_selection* foi utilizada a função *train_test_split* para dividir as bases de dados em treino e teste de forma estratificada, ou seja, mantendo a proporcionalidade dos rótulos.

Quatro scripts foram criados, dois para a base *wine* e dois para a base *default_features_1059_tracks*. Cada um desses arquivos possui a implementação da rede em uma ou duas camadas. A codificação da rede é a mesma em todos eles, o que muda é a plotagem dos resultados e a implementação em uma ou duas camadas. Acompanhado desses quatro scripts há também o script *nntest.py*. Esse script contém a implementação da rede de forma pura, ou seja, não plota resultados.

Treinamento e Resultados:

No caso da base *wine* a rede de duas camadas possui 20 neurônios na primeira camada escondida e 10 na segunda, enquanto a rede de uma camada possui 50. Para efeito

de comparação o número de ciclos/épocas é fixado em 10 mil, a taxa de aprendizado em 0.3 e momento em 0.95. As imagens a seguir mostram o desemprenho da rede em diferentes cenários.

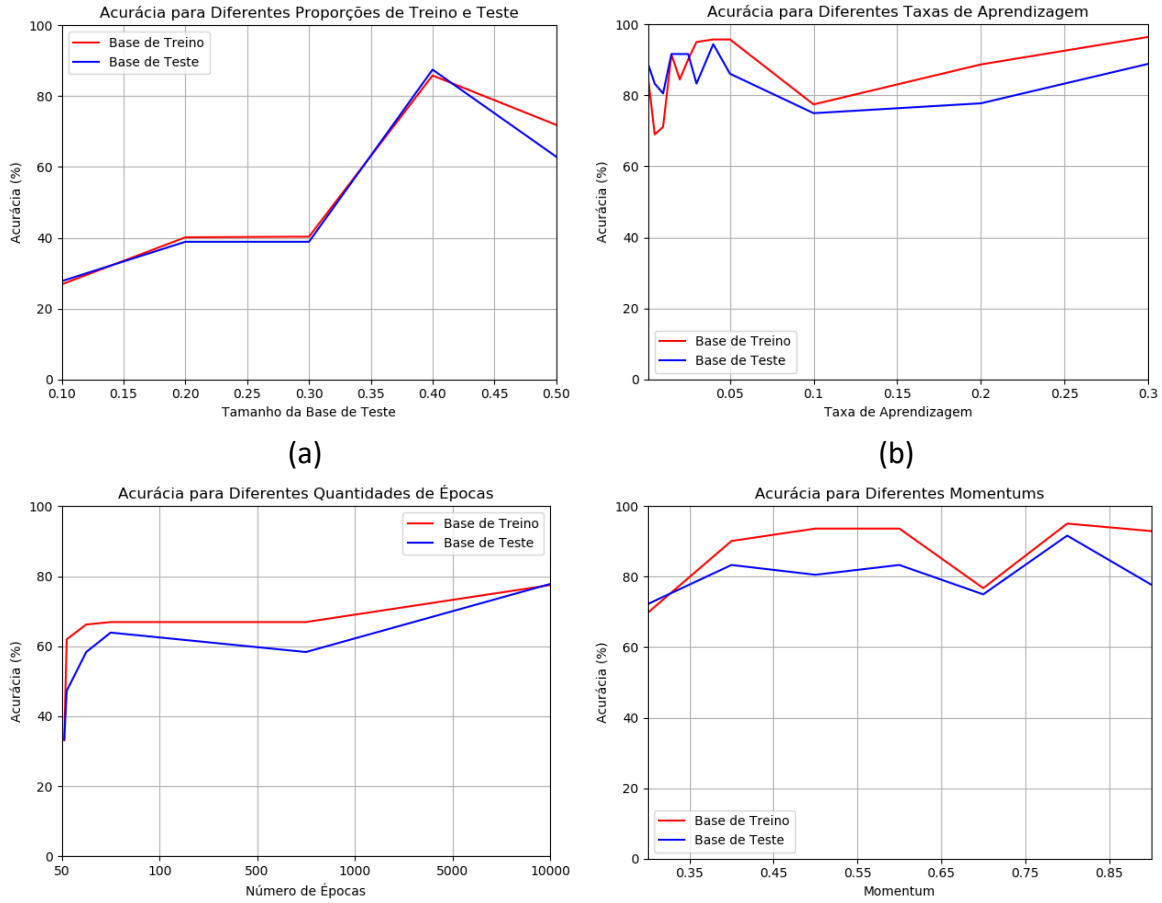
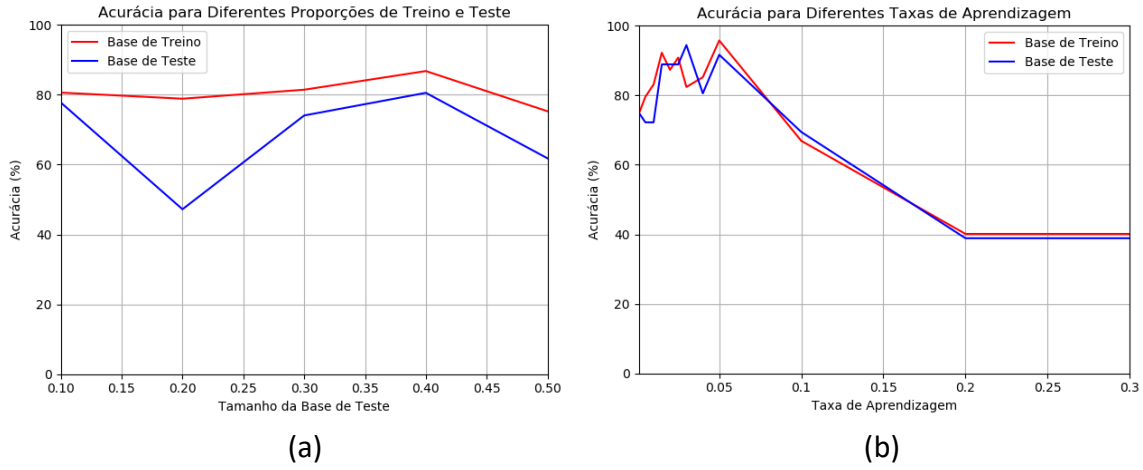
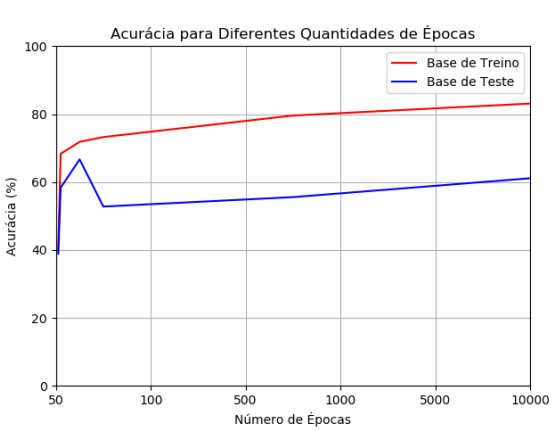
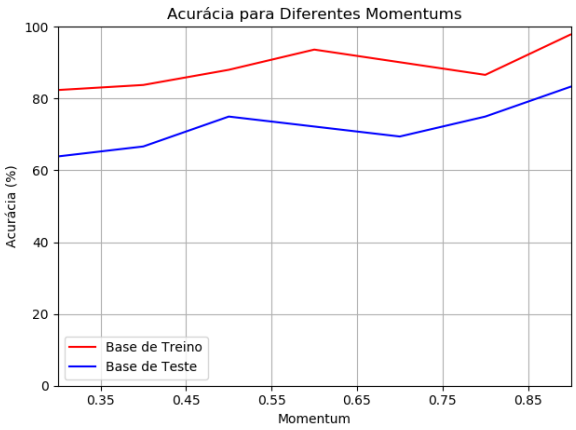


Figura 1: Desempenho da rede de *uma camada* em diferentes cenários para a base *wine*. (a) variando a proporção de treino e teste; (b) variando a taxa de aprendizagem. (c) variando a quantidade de épocas; e (d) variando o valor do momento.



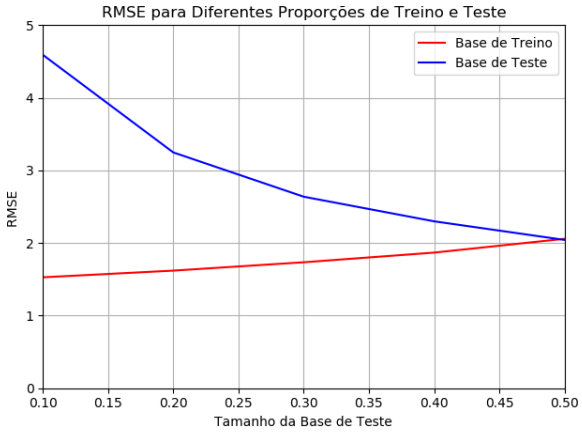


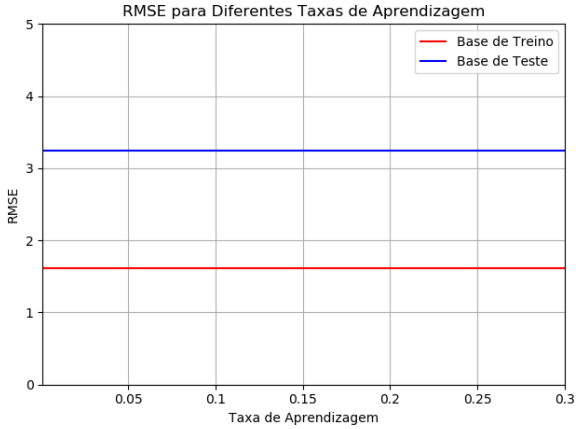


(c) (d)

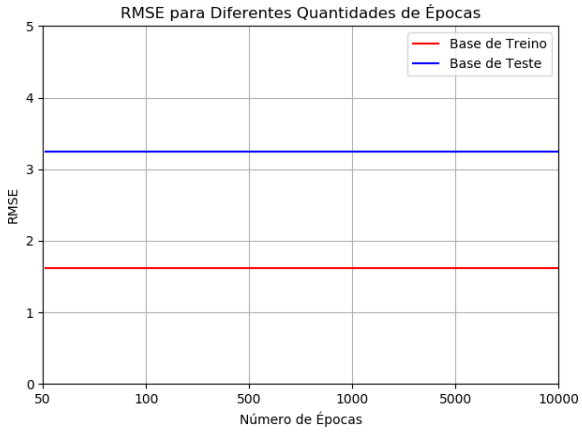
Figura 2: Desempenho da rede de *duas camadas* em diferentes cenários para a base *wine*. (a) variando a proporção de treino e teste; (b) variando a taxa de aprendizagem. (c) variando a quantidade de épocas; e (d) variando o valor do momento.

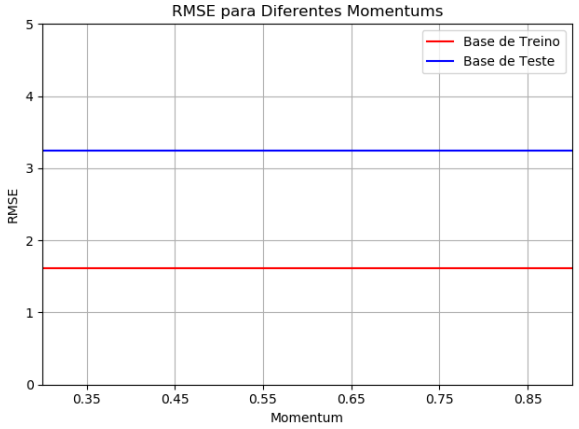
No caso da base *default_features_1059_tracks* a rede de duas camadas possui 70 neurônios na primeira camada escondida e 30 na segunda, enquanto a rede de uma camada possui 50. Para efeito de comparação, assim como na base *wine*, o número de ciclos/épocas é fixado em 10 mil, a taxa de aprendizado em 0.3 e momento em 0.95. As imagens a seguir mostram o desemprenho da rede em diferentes cenários.





(a) (b)





(c) (d)

Figura 3: Desempenho da rede de *uma camada* em diferentes cenários para a base *default_features_1059_tracks*. (a) variando a proporção de treino e teste; (b) variando a taxa de aprendizagem. (c) variando a quantidade de épocas; e (d) variando o valor do momento.

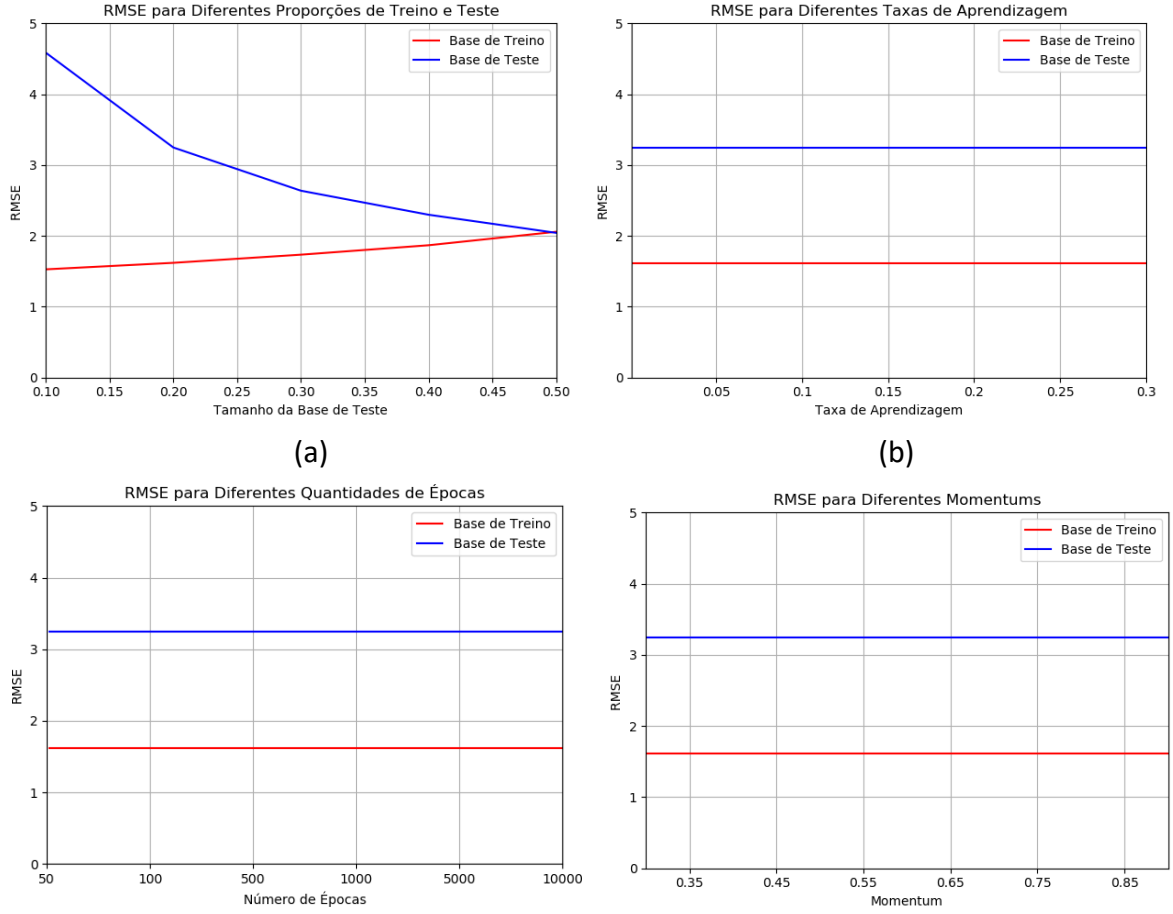


Figura 4: Desempenho da rede de *duas camadas* em diferentes cenários para a base *default_features_1059_tracks*. (a) variando a proporção de treino e teste; (b) variando a taxa de aprendizagem. (c) variando a quantidade de épocas; e (d) variando o valor do momento.

Conclusões:

Ao analisar a Figura 1 pode-se concluir que o aprendizado da rede é sensível a todos os parâmetros. Curiosamente a melhor divisão das bases de treino e teste é 60 e 40%, as melhores taxas de aprendizagem giram em torno de 0.05 e 0.25, o número de ciclos quanto maior melhor e o momento sofre uma pequena perda entre 0.65 e 0.75. Já o caso da rede de duas camadas, os resultados são um pouco diferentes. A melhor proporção de treino e teste ainda é 60 e 40%, o número de ciclos quanto maior melhor, e a performance do momento também cai quando este varia de 0.65 e 0.75. Porém a taxa de aprendizado tem um comportamento completamente diferente. Nessa configuração em duas camadas a melhor taxa de aprendizado gira em torno de 0.05. O gráfico indica também que quando a taxa é muito grande, o algoritmo não converge e tem resultados piores que o chute aleatório.

Para a base de dados *default_features_1059_tracks* os resultados não são promissores. A rede usada é exatamente a mesma da rede em *wine* porém o algoritmo não

consegue convergir independente de como os parâmetros são variados. No caso da proporção de treino as Figuras 3 e 4 (a) indicam que quanto maior o número de exemplos de treino melhor é a acurácia. Outro ponto a destacar é o fato do número de camadas também ser irrelevante para o algoritmo. Esse resultado indica duas possibilidades 1) a rede não está configurada para aplicações de regressão e algo a mais deve ser codificado; 2) os dados de entrada devem ser pré-processados de forma melhor pois a normalização apenas não é suficiente para resolver o problema.