Relatório - Exercício de Programação

Disciplina: ACH2016 Inteligência Artificial

Denis S. Shiroma[1], Laury Bueno[2], Lucas C. Delboni[3], Marcos V. de A. Giurni[4]

8598603[1], 5648179[2], 8516006[3], 8516080[4]

Escola de Artes Ciências e Humanidades (EACH)

Universidade de São Paulo (USP) - São Paulo, SP - Brasil, 2015.

1. Sobre a Implementação

a. Rede Neural *Multilayer Perceptron* (MLP)

A implementação da rede MLP teve como base o algoritmo apresentado pela Fausett (1993). Foi totalmente desenvolvido sem a utilização de códigos ou trechos de códigos de terceiros. Utilizando a linguagem Java, as duas principais classes são:

- Neuronio.java: contém toda a definição de um neurônio genérico, seja ele da camada de entrada, camada escondida ou de saída. Além das informações sobre seus pesos e vieses, é função de cada neurônio calcular o resultado de sua função de ativação. A classe também possui o método derivada(), que calcula a derivada da função de ativação bipolar sigmóide. O cálculo dessa derivada envolve uma simplificação apresentada por Fausett (1993): $\frac{(1+x)(1-x)}{2}$, em que x é o resultado da função de ativação.
- Rede.java: responsável por controlar todo o funcionamento da rede MLP juntamente com as demais classes. É nessa classe que são criadas as camadas de entrada, escondida e de saída. Além de ter todo o controle sobre método de treinamento *backpropagation,* a classe define quando serão salvas informações referentes a sua execução, embora a manipulação desses logs seja atribuída a outros arquivos. Essa classe tem dois modos de funcionamento: treinamento e execução. É possível salvar uma rede em disco para posteriormente ser aberta novamente.

A arquitetura da rede é formada por uma camada de entrada, uma camada escondida e uma de saída. A camada de entrada é constituída dos 64 neurônios que representam os conjuntos de *pixels* dos dados de entrada. Baseado no conhecimento de que poucos neurônios na camada escondida ocasionarão *under-fit* dos dados, enquanto muitos ocasionarão *over-fit*, foi verificada uma melhor configuração dessa camada com 16 neurônios. A camada de saída

conta com 10 neurônios, cada um representando um algarismo de 0 a 9. Para que a rede aponte uma resposta coerente para um certo dado, o neurônio respectivo o número representado por esse dado deve retornar o valor 1, enquanto todos os outros devem retornar o valor -1. Caso mais de um neurônio retorne 1 ou nenhum o retorne, a resposta não é válida.

A inicialização dos pesos dos neurônios pode ser aleatória ou todos eles definidos como zero. No caso de optar-se por instanciá-los como zero, o treinamento da rede tende a não progredir, já que o erro retropropagado da camada de saída à camada escondida também será zero e praticamente não haverá ajuste de pesos entre a camada de entrada e a camada escondida[1]. Isso levará o aprendizado a parar num ponto de mínimo local. A inicialização dos pesos de maneira aleatória minimiza as chances disso acontecer.[2]

Dado que uma taxa de aprendizado muito pequena fará com que a rede aprenda muito devagar, e uma taxa muito alta fará com que a rede oscile sobre um ponto de mínimo local, verificamos que uma boa taxa é a de 0.001.

b. Rede Neural *Learning Vector Quantization* (LVQ)

A implementação desta parte do trabalho foi desenvolvido, desde o início, levando em conta somente as descrições e o algoritmo apresentado pela Fausett (1993) e conceitos apresentados em classe, não possuindo então a utilização de código de terceiros. Assim nós particionamos todo o corpo da LVQ em três arquivos principais, sendo eles:

Inicializa.java: Arquivo contendo a classe Inicializa responsável por receber os dados de entrada para a etapa de treinamento e validação, atribuindo-os em matrizes (*arrays* 2d) que serão processados futuramente. Além de possuir o encargo de inicializar os vetores de pesos (neurônios de saída), conferindo um número de neurônio para cada classe que deve ser classificada. Tendo alguns métodos diferentes para essa inicialização (*PesosNulos, PesosRandom e PesosPrimeiraEntrada*).

LVQ.java: Contém a classe LVQ, podendo se dizer que é o núcleo do algoritmo, pois é ela que carregara neurônios de saída, e que executara o treinamento, validação e teste . Quando iniciada, deve receber uma classe Inicializa como parâmetro para conseguir iniciar os pesos, e logo após disso, já se pode fazer o treinamento. Além dos métodos principais dessa rede neural (Aprendizado, Teste e Validação), cada objeto LVQ possui

uma subclasse destina as condições de parada do treinamento. Ao final de ocorrer o aprendizado os vetores de pesos, já treinados, estarão prontos para serem testados, e podem ser manipulados livremente, pois se trata de um dos atributos dessa classe.

Treinamento.java: Possui a classe Treinamento que será chamada por um objeto LVQ que deseja executar o treinamento, possuindo também uma classe chamada OperacaoVetores que possui diversos métodos entre vetores, como: distância euclidiana, soma entre vetores, etc.

Um ponto interessante da implementação dessa rede neural, é que tanto para os dados de entrada e os vetores de pesos usamos matrizes do java como estrutura de dados base, pois suas características são as que mais se assemelham ao conceito de vetores. Além de podermos usufruir das facilidades dos índices, que auxiliaram bastante na construção de funções entre vetores.

Existem algumas características do sistema que foram prontamente criados e elaborados pelo grupo, assim possuindo aspectos únicos e que torna essa rede neural diferente de muitas outras. Um dos pontos que podem ser citados é a função que determina o momento no qual o algoritmo deve parar de rodar. Onde em intervalos de épocas (interações de aprendizado) ocorrem validações com dados de entrada não usados no treinamento, então verificamos a taxa de erro resultante dessas validações e comparamos com uma taxa de erro gerada sobre os próprios dados de treinamento. Para concluir, determinamos se durante as épocas vem havendo muitas pioras na taxa de erro sobre a validação, caso seja confirmado que essa taxa vem piorando muito, podemos parar o algoritmo e recuperar os neurônios com os melhores estados já alcançados.

Outro diferencial interessante que essa rede possui, é a maneira na qual os vetores de pesos são inicializados. Além de podermos escolher o método de inicialização dos valores, podemos modificar a própria estrutura de vetores de pesos, escolhendo quantos neurônios devem existir por cada classe, no entanto, foi definida uma regra que garante que todas as classes terão as mesmas quantidades de neurônios classificadores para si, evitando então que geremos estruturas falhas que permitem que alguma classe não tenha nenhum neurônio, e assim não havendo uma classificação para essa determinada espécie.

No entanto essa foi só uma passada rápida pelos principais aspectos da lvq, e muitos outros pontos interessantes podem ser encontrados no próprio código e comentários dela.

c. Pré-processamento

Para o pré-processamento, todos os códigos foram implementados pelo grupo, sem o uso de código de terceiros.

Foi feito um novo projeto “Pre-Processamento” que possui a classe “Input” que é usada para abrir os arquivos e transforma-los em uma matriz de doubles. A classe “Output” responsável por receber um arranjo de String e escrever em um arquivo. A classe “Normaliza” que possui a função “minMax” responsável por fazer a normalização MinMax e a função “zScore” responsável por normalizar usando a técnica z-score, essa classe também tem métodos para calcular a media e o desvio padrão de colunas para ajudar na normalização. O *main* do projeto está na classe “Main” que primeiro normaliza e, depois, cria os arquivos usando o *holdout*.

d. Procedimentos de avaliação do classificador

Na LVQ, os procedimentos usados para avaliar o classificador foi a matriz de confusão e informações taxa de erro do treinamento e da validação, caraterizada pela quantidade de erro sobre a quantidade total de dados de entrada, podendo ser obtido também a taxa de acerto que ao contrário da taxa de erro, leva em conta somente o número de classificações certas. A matriz de confusão está na classe “MatrizConfusao” que guarda uma matriz de inteiros contendo a matriz de confusão e algumas funções como “adicionaMatriz” que adiciona um em uma posição pré determinada na matriz de confusão, a “acurácia” e “erro” que retorna a acurácia e erro respectivamente da matriz de confusão.

Como condição de parada da LVQ, foi usado: a taxa de erro obtido a partir de dados de validação, e um contador de pioras que ocorriam nesse índice, e caso chegassem no limite de degradações permitidas, se encerrava o processo.

Na MLP, também foi usada a acurácia para medir o desempenho em treinamento, validação e teste. O erro quadrado não foi utilizado neste relatório, porque o grupo encontrou problemas para interpreta-lo. Nos testes realizados, essa medida subia mesmo quando a taxa de erros caía. Dessa forma, usar o erro quadrado como condição de parada sempre levou a redes pouco e mal treinadas.

2. Sobre o pré-processamento

A única decisão tomada para exclusão de atributos foi deletar os atributos que possuíam desvio padrão igual a zero. Pois, como o atributo não muda absolutamente nada, ele não oferece informação nenhuma para as redes neurais. Apenas 2 atributos foram encontrados com essa restrição e deletados.

O grupo discutiu a possibilidade de deletar também os atributos que tinham o desvio padrão muito pequeno (quando 90% dos elementos de uma coluna são iguais, por exemplo). Porém, em uma análise posterior, percebeu-se que esses atributos poderiam ajudar a rede neural, pois eles só são diferentes quando pertencem a uma determinada classe.

Foram utilizados as técnicas de normalização z-score e MinMax. As duas normalizações mantém a distribuição original dos dados. A MinMax é uma técnica linear. Z-score converte os dados para uma distribuição de media 0 e desvio padrão 1[1], assim como a distribuição gausiana (distribuição normal).

Sobre o range do MinMax, foi escolhido de 0.1 a 0.9, pois esse range previne que os pesos dos neurônios continuem os mesmos (Qualquer número multiplicado por 0 é igual a 0 e qualquer número multiplicado por 1 é igual ao número). Esse domínio foi proposto por Basheer & Najmeer (2000)[2]. O z-score, não tem definido o domínio e vária mais que o MinMax.

Era esperado que, usando a normalização z-score, as redes neurais tenham o melhor resultado, porém, após a análise dos resultados, pode-se concluir que o z-score não foi positivo em todos os conjunto de dados.

Os testes das normalizações foram feitos tanto para a LVQ quanto pra MLP. Na LVQ foi feito o pior caso e o melhor caso para ver o quanto as normalizações ajudaram. No pior caso, foi gerado os seguintes casos.

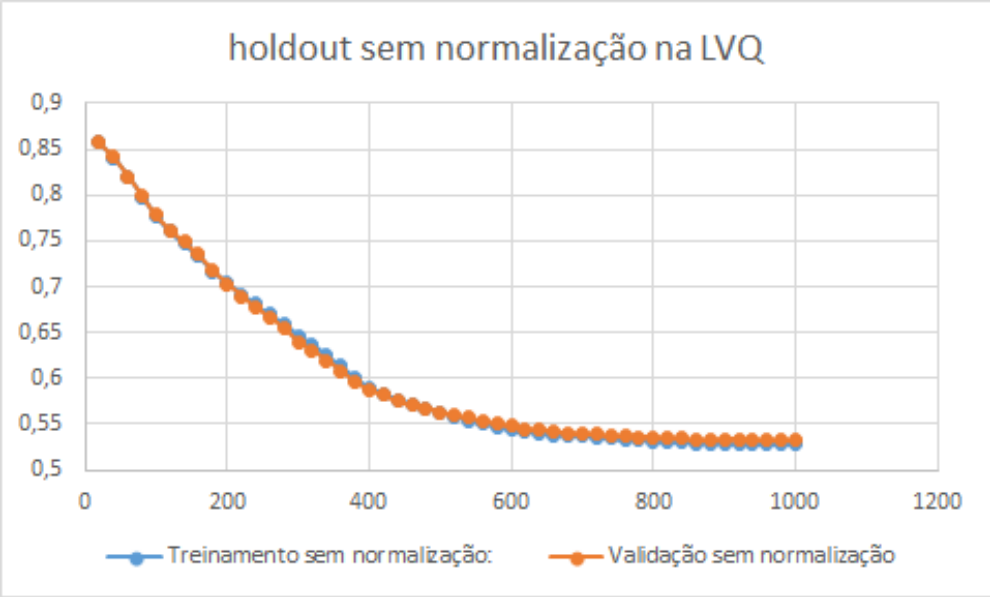

Gráfico 1. taxa de erro sobre o teste e validação no teste com os pesos dos neurônios iniciando aleatoriamente, 1 neurônio para cada classe, taxa de aprendizado de 0,000001 e redução da taxa de aprendizado de 0.000000001 na LVQ com o conjunto de dados não normalizados.

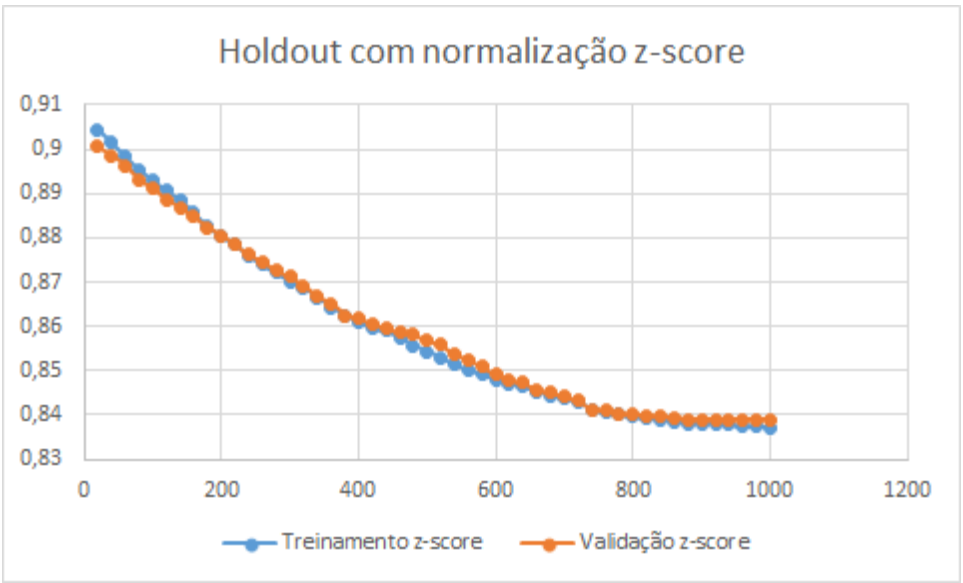

Gráfico 2. taxa de erro sobre o teste e validação no teste com os pesos dos neurônios iniciando aleatoriamente, 1 neurônio para cada classe, taxa de aprendizado de 0,000001 e redução da taxa de aprendizado de 0.000000001 na LVQ com conjunto de dados normalizados usando a técnica do MinMax.

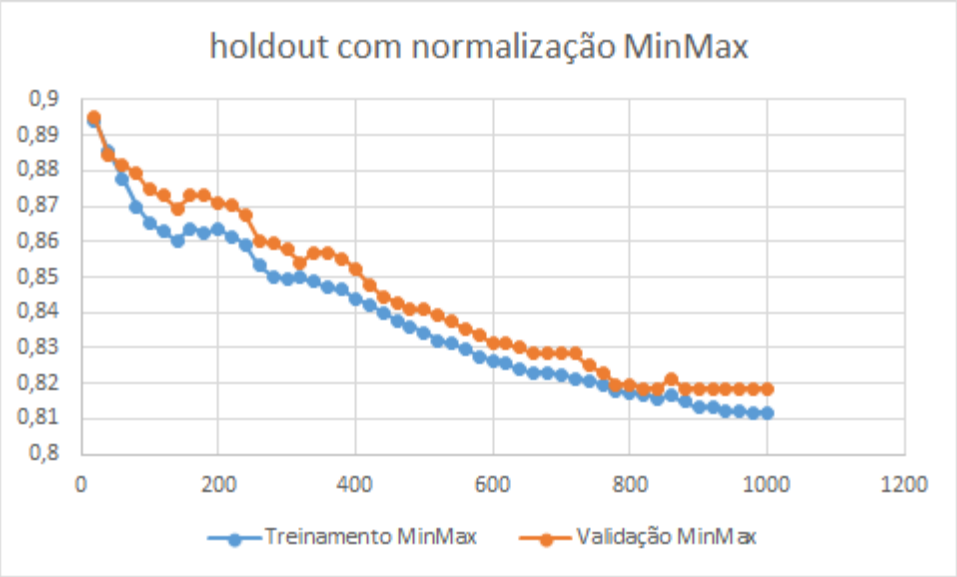

Gráfico 3. taxa de erro sobre o teste e validação no teste com os pesos dos neurônios iniciando aleatoriamente, 1 neurônio para cada classe, taxa de aprendizado de 0,000001 e redução da taxa de aprendizado de 0.000000001 na LVQ com conjunto de dados normalizados usando a técnica do z-score.

Com os gráficos gerados, pode-se observar que o uso da normalização atrapalhou a rede. Mesmo com MinMax, que obteve o melhor resultado das normalizações, pode-se observar uma grande piora com o uso da normalização.

Quando se analisa o melhor caso, se obtém o seguinte gráficos.

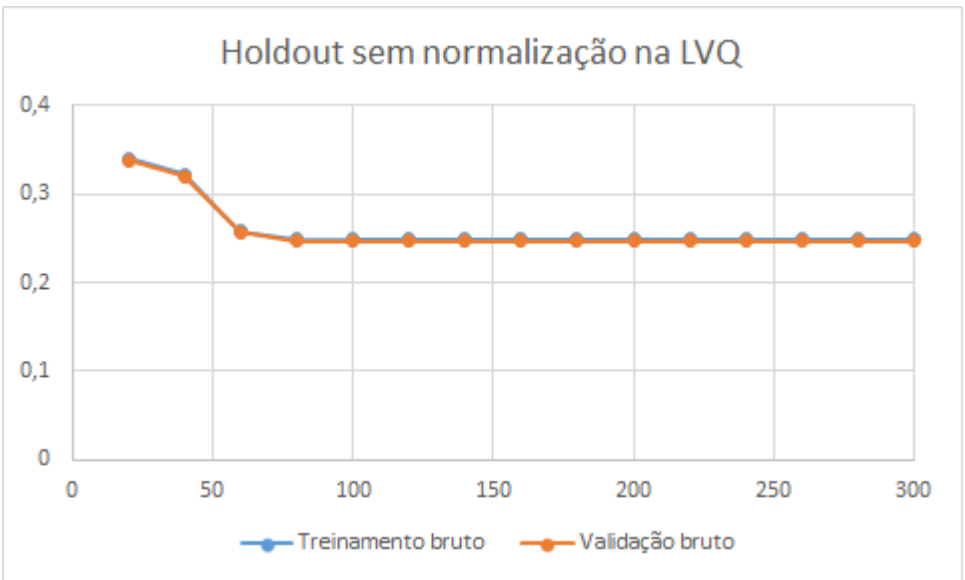

Gráfico 4. taxa de erro sobre o teste e validação no teste com os pesos dos neurônios iniciando com 0, 5 neurônios para cada classe, taxa de aprendizado de 0.0001 e redução da taxa de aprendizado de 0.0000001 na LVQ com conjunto de dados não normalizados.

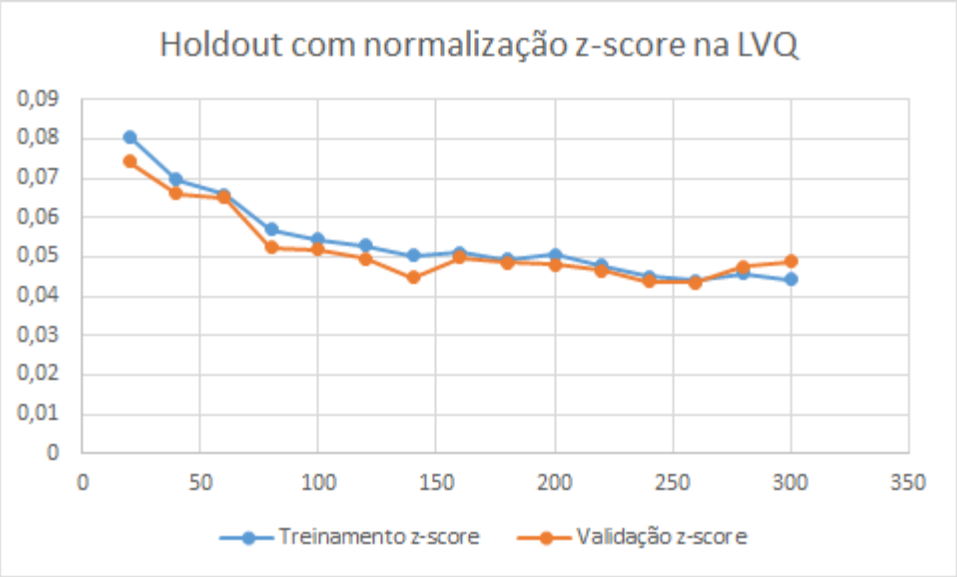

Gráfico 5. taxa de erro sobre o teste e validação no teste com os pesos dos neurônios iniciando com 0, 5 neurônios para cada classe, taxa de aprendizado de 0.0001 e redução da taxa de aprendizado de 0.0000001 na LVQ com conjunto de dados normalizado usando a técnica z-score.

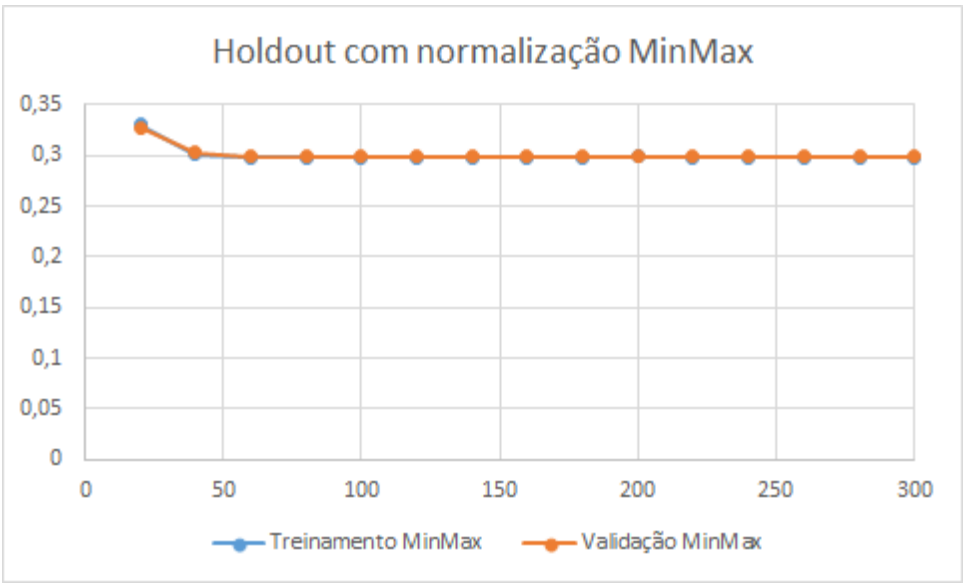

Gráfico 6. taxa de erro sobre o teste e validação no teste com os pesos dos neurônios iniciando com 0, 5 neurônios para cada classe, taxa de aprendizado de 0.0001 e redução da taxa de aprendizado de 0.0000001 na LVQ com conjunto de dados normalizado usando a técnica MinMax.

Com eles, foi observado que a normalização ajudou bastante no aprendizado da rede neural. Pricipalmente com z-score que a rede começou melhor do que terminou a rede sem normalização.

Para uma melhor análise do melhor caso, foram geradas as matrizes de confusão e feito a média delas do teste realizado no gráfico 4 e 5. Obtendo assim as tabelas 1 e 2.

	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
0.0	111.1	0.0	0.0	0.0	0.7	0.0	0.2	0.0	0.0	0.0
1.0	0.0	102.1	4.6	0.0	0.1	0.3	1.6	0.8	0.0	2.5
2.0	1.4	1.3	107.8	0.3	0.0	0.0	0.1	1.6	0.0	0.5
3.0	2.5	5.2	9.8	64.8	0.2	8.8	0.4	8.5	0.0	12.8
4.0	1.9	4.3	0.5	0.0	93.0	1.6	2.6	8.3	0.0	0.8
5.0	0.5	0.0	0.7	0.2	1.0	104.5	0.6	0.0	0.0	5.5
6.0	0.2	1.4	0.0	0.0	0.6	0.0	109.8	0.0	0.0	0.0
7.0	0.0	0.3	0.2	0.0	0.4	0.6	0.0	110.5	0.0	0.0
8.0	6.2	28.0	14.3	8.5	4.5	19.8	15.2	9.7	0.0	5.8
9.0	13.2	6.2	1.4	17.4	6.9	10.4	0.5	7.8	0.0	48.2

Tabela 1. Média das matrizes de confusão geradas pelo teste do gráfico 4

	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
0.0	111.1	0.0	0.1	0.1	0.1	0.1	0.4	0.0	0.0	0.1
1.0	0.0	105.9	0.6	0.2	0.1	0.2	0.7	0.1	2.3	1.9
2.0	0.2	2.2	107.8	0.8	0.3	0.0	0.0	0.1	1.3	0.3
3.0	0.0	0.1	1.4	105.3	0.0	1.1	0.0	1.3	1.6	2.2
4.0	0.5	2.0	0.0	0.0	103.5	0.0	0.9	3.4	0.1	2.6
5.0	0.1	0.6	0.1	1.2	0.2	102.5	0.4	1.2	1.2	5.5
6.0	0.3	0.8	0.0	0.0	0.1	0.4	110.2	0.1	0.1	0.0

7.0	0.0	0.3	0.4	0.7	0.2	0.4	0.0	107.4	0.2	2.4
8.0	0.0	4.7	0.4	0.5	0.7	0.2	0.6	0.0	103.0	1.9
9.0	0.2	1.4	0.2	2.2	0.7	1.9	0.2	1.2	2.0	102.0

Tabela 2. Média das matrizes de confusão geradas pelo teste do gráfico 5.

Com essas tabelas, pode-se observar que, usando a normalização z-score, a LVQ passou a aprender a classe 8 (que antes ela não teve nenhum acerto).

Na MLP foi feito o teste do que se esperava ser o melhor caso com os dados brutos. Foram encontrados os seguintes resultados.

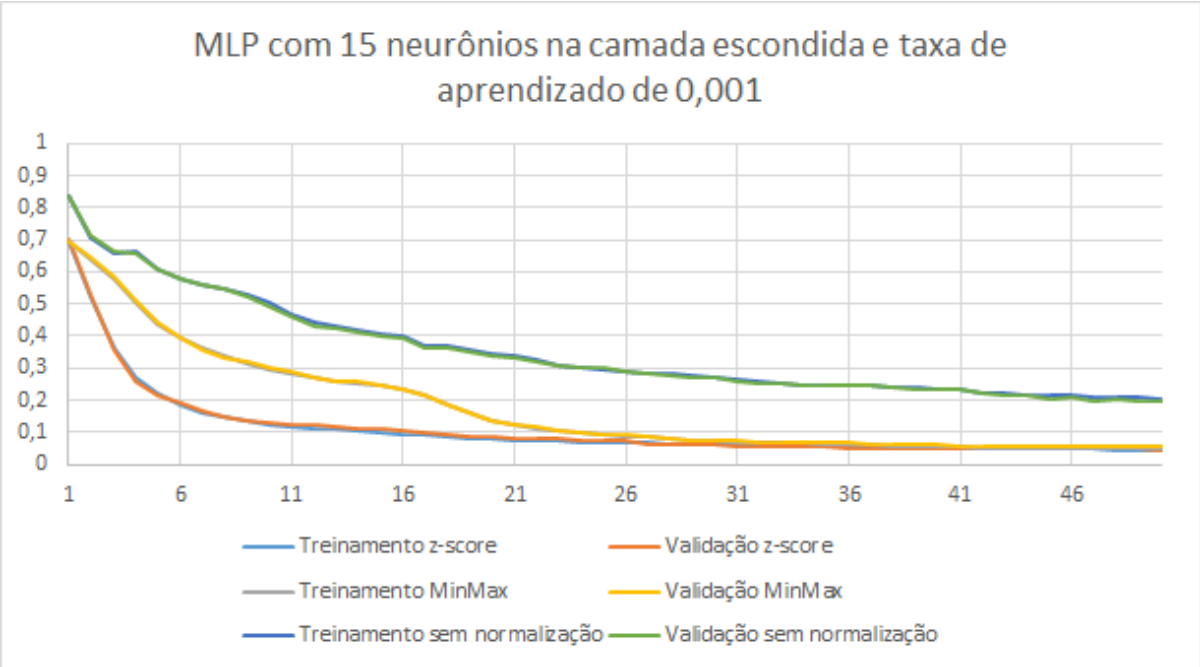

Gráfico 7. taxa de erro sobre o teste e validação no teste com os pesos dos neurônios iniciando randomicamente, 15 neurônios na camada escondida, taxa de aprendizado de 0.001 na MLP.

Com esse gráfico, pode-se observar que, entre as épocas 1 e 26, usando a normalização z-score, a rede neural possui uma melhora substancial em comparação com a normalização MinMax e sem normalização. Porém, após a época 26, tanto a normalização MinMax quanto a z-score possuem a taxa de erro bem próximas, mas, ainda assim, melhores do que sem a normalização. Mostrando que, para esse caso, a técnica do z-score aprende mais rápido e se sai melhor do que sem normalização.

A análise feita pelo grupo, permite concluir que a técnica z-score, apesar de ser mais custosa em termos de implementação, execução e dificultar o aprendizado no pior caso , é benéfica para a rede neural, pois, no melhor caso, ela permite que a rede aprenda mais rápido e sua taxa de erro no final é bem semelhante à normalização MinMax.

Foi gerada a média das matrizes de confusão para os 3 casos separadamente do teste feito no gráfico 7.

	0	1	2	3	4	5	6	7	8	9
0	106.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
2	0.0	0.0	105.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	103.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	102.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	101.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	105.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	102.0	0.0	0.0
8	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	95.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	97.0

Tabela 3. Média das matrizes de confusão com o conjunto de dados não normalizados geradas pelo teste do gráfico 7.

Foi analisado também as outras matrizes de confusão, porém todas elas estavam semelhantes. Para algumas classes, com os dados não normalizados, teve mais acerto do que com os dados normalizados (tanto com z-score quanto MinMax), porém para a classe que se apresentou ser mais difícil de ser aprendida (classe 8), normalizado com z-score se saiu melhor.
Não foi feito busca de *outliers*.

3. Divisão dos conjuntos

Para criar os conjuntos de dados de treino, validação e teste foi utilizada a técnica *holdout* que se consistiu em pegar o conjunto de dados geral e embaralhar ele para que possa ter diferentes conjuntos de dados, logo após, foi feito 3 arquivos (60% do conjunto de dados para treino, 20% para validação e outros 20% para teste) de forma que nenhum dado se repita e que a distribuição de classes seja igualitária na medida do possível (o conjunto de dados total possui mais dados de algumas classes).

Esse algoritmo foi executado 10 vezes, gerando, assim, 10 arquivos de teste, treino e validação. Esses mesmo conjuntos foram testados na LVQ e na MLP.

Foi discutido sobre a possibilidade de usar a técnica *k-fold* para se ter uma medida mais confiável, porém , como o *holdout* seria executado 10 vezes, foi escolhido apenas fazer o *holdout*. O *k-fold* consiste em dividir o conjunto de dados em k conjuntos. Em cada execução da rede neural, seria utilizado um desses conjuntos para treino e os outros conjuntos seriam utilizados para teste.

Na fase de treino, foi utilizado o conjunto de dados treino para treinar e outro conjunto de dados de validação para dificultar que a rede decore o treinamento criando, assim, uma falsa impressão de aprendizado. Depois do treino, a rede neural foi testada usando o conjunto de dados de teste.

Foi feito também a media e o desvio padrão das matrizes de confusão para analisar o quanto a rede neural muda de acordo com o conjunto de dados do treino e da validação e a media de acertos e erros com os diferentes conjuntos feitos pelo *holdout*.

O *holdout* foi implementado na função “holdout” da classe “Input” dentro do projeto “Pre-Processamento e foi todo implementado pelo grupo sem o uso de código de terceiros.

4. Descrição da sintonização de parâmetros

a. Rede Neural *Multi Layer Perceptron* (MLP)

As principais características estruturais que influenciam o comportamento de uma rede MLP são a taxa de aprendizado, a quantidade de neurônios na camada escondida, a função de ativação e o pré-processamento de dados. Diversas dessas variáveis afetam também a LVQ, como já foi explicitado, mas sua influência pode ser diferente em um perceptron.

Todos os testes desta seção foram realizados com uma tolerância de 2 mil épocas à perda de acurácia em validação. Ou seja, todas as redes apresentadas a seguir tiveram 2 mil épocas a

mais de treinamento depois de terem obtido sua última melhora. Naturalmente, a versão da rede aplicada no conjunto de testes sempre é a melhor encontrada durante todo o processo.

a.1. Função de ativação

Na maioria dos testes apresentados neste relatório para a MLP, foi usada a bipolar sigmóide como função de ativação. Mas, para comentar seu comportamento, testa-se também a binária sigmóide. Os testes a seguir foram feitos com 15 neurônios na camada escondida e taxa de aprendizado fixa em 0,001.

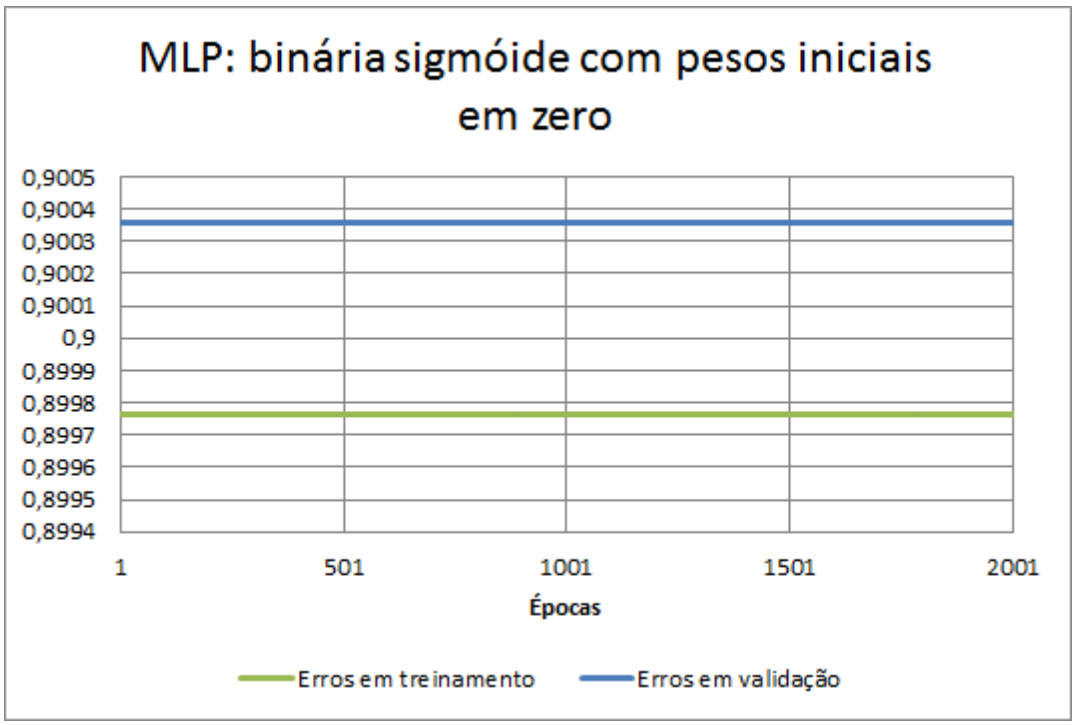

Gráfico 8.

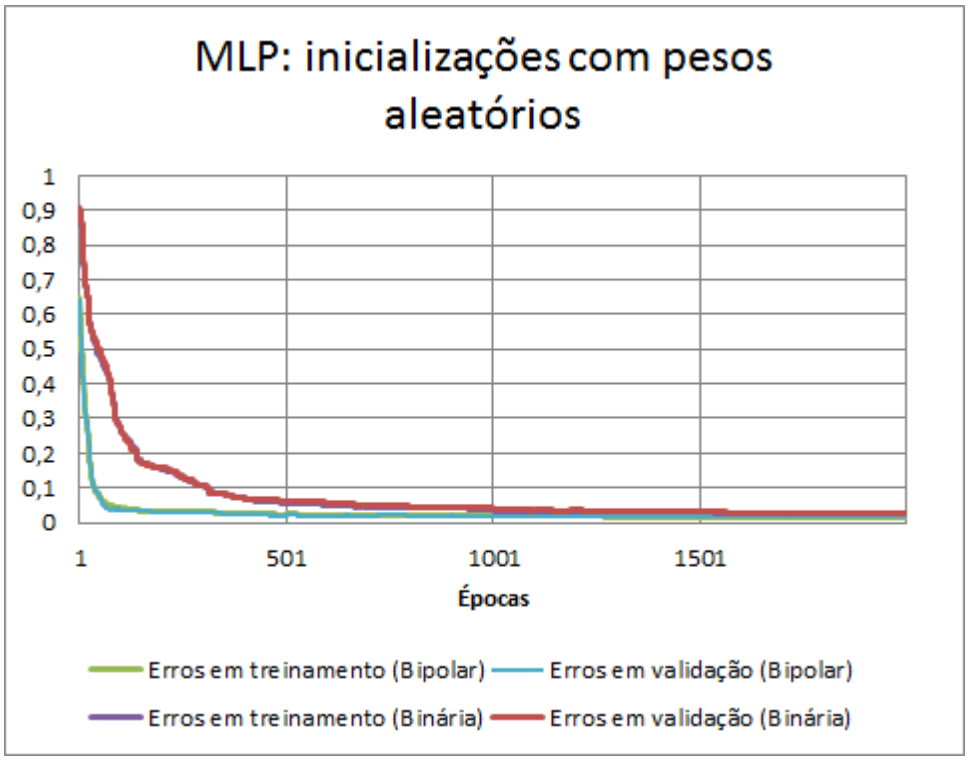

Gráfico 9.

É possível notar, de imediato, que a inicialização de pesos em "zero" prejudica o aprendizado quando a função de ativação utilizada é a binária sigmóide. O mesmo ocorre quando a bipolar é usada nesse contexto, mas com consequências piores: quase todos os pesos da rede ficam permanentemente paralisados em "zero" independentemente do tempo de execução. Isso ocorre porque os pesos iniciais acabam anulando quase todos os fatores de variação de peso da etapa de Back Propagation. Os únicos pesos que conseguem variar são os vieses da camada escondida. Com a função binária, a alteração de pesos é realizada, mas o aprendizado acaba anulado e não há melhoria de desempenho.

Segundo ROJAS, R. (1996, p97), quando os pesos iniciais selecionados são muito próximos de zero (ou zero), o fator de correção propagado tende a também ser muito próximo de zero (ou zero). Na prática, isso retarda ou pode inibir completamente o aprendizado de uma MLP. Essa proposta encontra respaldo nos dados apresentados.

a.2. Número de neurônios na camada escondida

Foi verificado nas literaturas que o número ideal de neurônios a serem utilizados na camada escondida é objeto de grande debate. Há quem defenda que apenas 2 neurônios é o suficiente, porém verificamos que a rede com apenas 2 neurônios na camada escondida causa *under-fit*.

Esse comportamento ocorre porque os poucos neurônios utilizados não são suficientes para "alimentar" a função de ativação devido à complexidade dos dados de entrada. Da mesma forma, um grande número de neurônios na camada escondida faria com que houvessem mais pesos para serem modificados, fazendo com que a rede tornasse complexa demais para lidar com os dados, causando *over-fit*. O mais aceito é que não há uma forma de definir um número ideal de neurônios, embora seja conveniente escolher um valor que esteja entre o número de neurônios da camada de entrada e o número de neurônios da camada de saída, de modo que esse número não seja tão baixo (para não causar *under-fit*) e nem tão alto (para não causar *over-fit*).

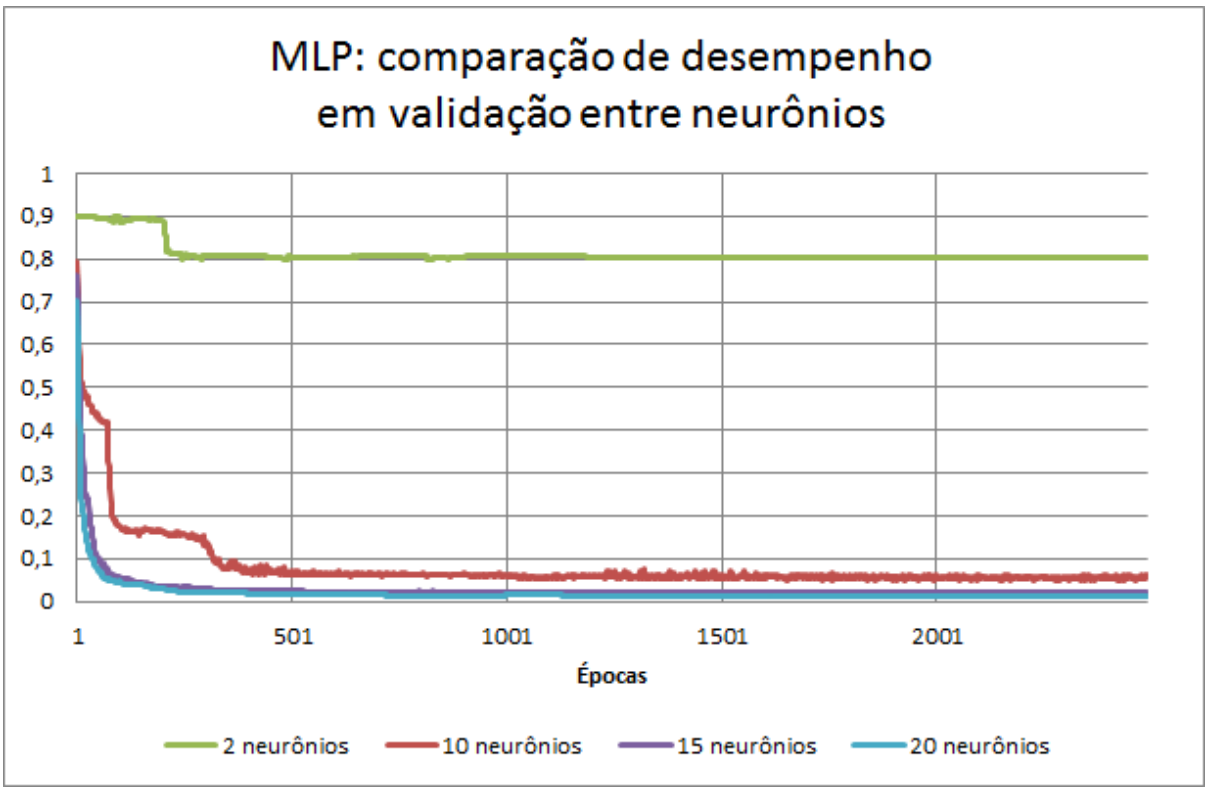

Gráfico 10. Quanto mais neurônios temos na camada escondida, melhor foi o desempenho alcançado

a.3. Taxa de aprendizado

Realizamos testes com a taxa de aprendizado sendo reduzida gradativamente e com ela fixa em 0,001. Utilizando a função de ativação bipolar sigmóide, a taxa de aprendizado fixa apresentou melhores resultados, uma vez que, na redução gradativa, a grande taxa de aprendizado inicial fizesse com que a função variasse até um ponto em que, mesmo com a taxa de aprendizado assumindo valores baixos, não era mais possível encontrar um bom

ponto de solução. Quando a binária sigmóide foi utilizada como função de ativação, a técnica de reduzir gradativamente a taxa de aprendizado demonstrou melhores resultados em relação a mantê-la fixa, uma vez que com menores valores de taxa de aprendizado, a rede conseguiu menores taxas de erro.

b. Rede Neural *Learning Vector Quantization* (LVQ)

Existem diversos parâmetros que o algoritmo deve receber para executar seu processamento, dentre eles existem alguns que irão descrever e moldar a cara da arquitetura da lvq, assim se pode criar inúmeras arquiteturas baseando nas diversas combinações dessas variáveis recebidas.

A tabela abaixo descreve algumas dessas arquiteturas que foram consideradas interessantes, e que logo serão usadas para se entender qual o resultado das mudanças de seus atributos.Vale ressaltar que todo o desenvolvimento das redes apresentadas aqui usou o método de *holdout* em sua construção.

Casos	Inicialização dos neurônios de saída	Nº Neuronino de Saída	taxa de Aprendizado	Redução taxa de aprendizado	número máximo de piores permitidas	Épocas Gastas em média	Acurácia Média sobre teste (taxa de Erro)
A	primeiraEntrada	1	0.01	0.00001	1	148.0	0.08852313167259784
B	primeiraEntrada	1	0.0001	0.0000001	1	234.0	0.09644128113878998
C	primeiraEntrada	1	0.000001	0,00000001	1	634.0	0.3145907473309609
D	primeiraEntrada	1	0.01	0.00001	3	762.0	0.08754448398576513
E	primeiraEntrada	5	0.01	0.00001	1	180.0	0.09051601423487546
F	primeiraEntrada	10	0.01	0.00001	1	78.0	0.023220640569395008
G	primeiraEntrada	10	0.01	0.00001	3	894.0	0.022597864768683265
H	nulo	1	0.0001	0.0000001	1	488.0	0.27918149466192155
I	nulo	1	0.0001	0.0000001	3	1000.0	0.23265124555160133
J	nulo	5	0.000001	0.000000001	3	1000.0	0.3956405693950177
K	nulo	10	0.01	0.00001	3	1000.0	0.23193950177935943
L	nulo	10	0.000001	0.000000001	3	1000.0	0.42615658362989295
M	aleatorio	10	0.01	0.00001	3	924.0	0.2238434163701067

Tabela A. com os casos que serão estudados. Para visualizar tabela completa veja AnexoA.

Para um rápido entendimento sobre a tabela acima, deve se levar em conta que cada coluna da tabela define um parâmetro que descreve a arquitetura da rede neural, e que cada linha declara uma sintonização de rede neural, ou seja, uma arquitetura lvq. Assim para a coluna "Inicialização" temos o tipo de inicialização dos vetores de pesos da nossa rede, podendo ser do tipo:

- Aleatório: definimos que os valores iniciais dos neurônios serão aleatórios, mas restringidos em um intervalo que vai de -1 até 1.
- Primeira Entrada: onde temos os valores inicias dos neurônios iguais aos primeiros valores de dados de treinamento, se levando em conta também as classes desses dados, tentando manter assim que alguma classe fique sem seu neurônio corresponde

de classificação. Pode-se observar que essa em relação as outras concede a lvq um avanço no aprendizado absurdo.

- Nulo: Esse argumento garante que os valores dos neurônios sejam incialmente zerados.

Já a segunda coluna da tabela descreve a quantidade de neurônios por classe, a terceira coluna correspondente à taxa de aprendizado que se refere ao valor inicial da taxa de aprendizado, sendo a redução da taxa de aprendizado um valor que irá ser subtraído da taxa de aprendizado durante as épocas.

Outro parâmetro interessante é o número máximo de pioras permitidas. Que assim poderá limitar quantas épocas será executado a depender da quantidade de vezes que houve uma regressão no resultado sobre as validações, e caso houve uma parada por esse motivo, o neurônio de melhor estado dentro do histórico de execução será recuperado, e será mantido como resposta final para o treinamento. Vale ressaltar que caso a taxa de aprendiza chegue à zero, a execução será parada automaticamente, e que irá se executar o mesmo processo de recuperação do vetor de pesos de melhor condição.

Agora as duas últimas colunas estão mais relacionadas, aos desempenhos e resultados de todo o processo de treinamento, validação e teste. Pois assim temos os dados médios da quantidade de épocas que conseguiu se realizar e a taxa de erro, levando em conta que essas médias foram tiradas através da execução de realizada 10 vezes da mesma arquitetura seguindo a estratégia de *holdout*.

b.2. Estudos sobre casos interessantes

Nessa parte do relatório, ira se iniciar estudos onde se observou comportamentos interessantes e que ilustram bem, o comportamento da rede neural. Daqui por diante quando referenciado uma arquitetura irá se usar o índice de casos da tabela “A” descrita anteriormente.

b.2.1. Comportamentos Interessantes segundo a taxa de aprendizado e seu fator de redução.

Dentre as sintonizações descritas na tabela, podemos observar que um dos atributos é a taxa de aprendizado, e com algumas análises consegue se observar características interessantes

que esse atributo causa no treinamento. Assim podemos fazer estudos em cimas de casos que foram interessantes, e que demonstram o peso desse atributo.

Levando em conta casos como o B e C que possuem praticamente os mesmo parâmetros, diferenciando somente nos atributos relacionados à taxa de aprendizado, conseguem se perceber que há uma clara diferença nos índices de aprendizado, e que devido ao baixo valor da taxa de aprendizado do caso C, o algoritmo aprendeu em uma velocidade muito degradante como resultado.

Consequentemente, devido ao valor mais alto da taxa de aprendizado do caso B temos que a rede aprendeu muito rápido. Outro ponto interessante a se considerar é que quanto mais alto a taxa de aprendizado é mais variações no ritmo de aprendizado existe, pois etapa de aproximação dos pesos caso o aprendizado seja muito alto, ele pode acabar errando o alvo, e por consequência, acaba se afastando do resultado, devido a essas considerações pode se levar em conta o número de épocas que as redes B e C conseguiram chegar.

Como a rede C tem um aprendizado muito baixo, a variação no ritmo de aprendizado é muito baixo também, assim existem menos pioras, pois diminui a chance do neurônio errar o alvo na aproximação, então a rede acaba chegando a um número de épocas superior ao da rede B. No entanto, mesmo que a rede B tenha menos épocas ela aprendeu consideravelmente muito mais que a rede C, assim quando calibrada direito essa taxa de aprendizado maximiza muito o processo de aprendizado.

Assim, podemos observar melhor a diferença que ocorreu na classificação, conforme essa variação que existe na taxa de aprendizado, entre a rede B e C.

-1.0	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
0.0	111.3	0.0	0.0	0.0	0.4	0.0	0.2	0.0	0.0	0.1
1.0	0.0	88.7	5.0	1.0	2.4	0.3	2.2	1.9	4.5	6.0
2.0	0.4	1.0	104.5	1.6	0.0	0.0	0.1	1.1	2.7	1.6
3.0	0.0	0.2	0.8	104.2	0.0	1.1	0.0	1.1	3.3	2.3
4.0	0.0	2.8	0.0	0.0	101.7	0.7	0.7	4.6	2.2	0.3
5.0	0.1	0.0	0.1	1.3	0.7	95.7	0.5	0.0	0.0	14.6
6.0	0.1	1.2	0.1	0.0	0.5	0.0	109.6	0.0	0.5	0.0
7.0	0.0	0.3	0.1	0.1	0.0	0.7	0.0	109.6	1.0	0.2
8.0	0.0	5.7	0.4	1.5	1.0	1.3	1.5	0.4	98.1	2.1
9.0	0.3	2.2	0.1	2.7	4.4	4.2	0.0	4.3	1.6	92.2

Matriz de Confusão segundo a média de classificação (caso B). Linhas representam a resposta esperada e colunas representam a respostada dada pela rede.

-1.0	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
0.0	109.2	0.1	0.0	0.1	0.6	0.6	0.4	0.0	0.1	0.9
1.0	1.6	62.3	13.0	4.6	5.0	5.0	4.7	4.5	5.3	6.0
2.0	2.3	15.9	66.0	12.3	0.1	0.8	1.9	3.1	5.1	5.5
3.0	0.3	2.0	2.9	81.6	0.0	3.4	0.2	2.1	4.9	15.6
4.0	3.0	7.2	0.2	0.0	85.7	2.7	5.4	5.8	2.0	1.0
5.0	3.1	1.9	2.5	5.9	1.5	67.4	2.9	1.6	7.0	19.2
6.0	8.0	0.9	0.2	0.1	2.0	0.8	98.3	0.0	1.6	0.1
7.0	0.0	4.2	6.1	2.3	6.9	0.9	0.1	88.2	2.3	1.0
8.0	3.5	14.7	9.0	8.1	1.2	6.1	5.8	2.4	53.2	8.0
9.0	3.5	1.9	1.5	12.4	4.1	15.1	0.3	7.5	7.2	58.5

Matriz de Confusão segundo a média de classificação (caso C). Linhas representam a resposta esperada e colunas representam a respostada dada pela rede.

Assim notamos uma boa diferença entre as duas, verificando que no caso B ele acerta bem mais que no caso C, independente se no B ele tenha rodado menos vezes.

Uma observação à parte, é que durante a fase de teste do trabalho, quando se usou uma taxa de aprendizado muito alta (taxas de aprendizados próximo ao valor 1), em maioria dos casos houveram pesos que acabaram se dispersando muito, chegando a estourar as limitações do java, e isso degradou totalmente o processamento do algoritmo.

b.2.2.Comportamentos Interessantes segundo a taxa de erro como critério de parada.

Detalharemos nesse caso mais particularidades interessantes sobre a rede lvq, utilizando dessa vez o caso A comparando com a rede D que possui basicamente os mesmo parâmetros que D, e que só se diferencia pela quantidade de pioras permitidas.

Como observado na tabela, mesmo com o aumento de pioras permitidas em A, não verificamos que houve uma grande melhora em relação a D. Pois o que verificamos em muito dos casos quando começa a haver pioras na taxa de erro, é que a rede pode estar entrando em uma fase onde irá haver muito distúrbios na taxa de erro, sem haver um grande avanço no aprendizado, pois a LVQ neste ponto começa a chegar ao seu limite, ou a rede começa se especializar em cima do treinamento, o que causa um vício, e consequentemente uma degradação em cima da validação.

Assim, quando limitamos o número de pioras em um em alguns casos estamos restringindo a rede de entrar nos momentos descritos acima, assim evitando processamentos desnecessários, podemos verificar isso melhor, com a comparação das redes Z e W pelos gráficos abaixo:

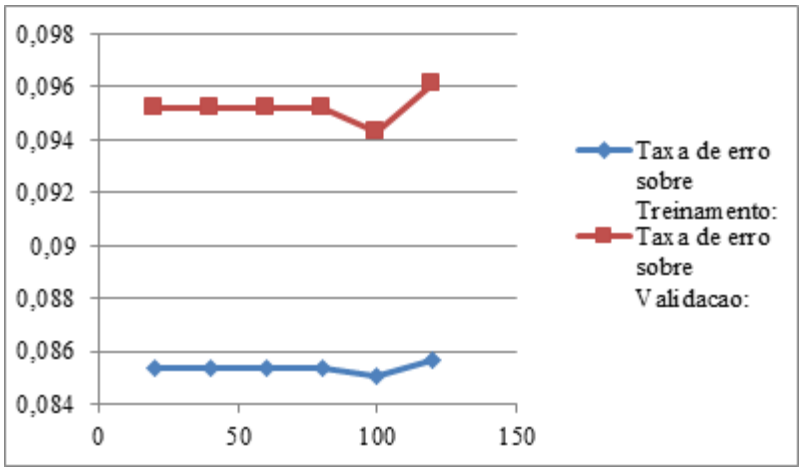

Gráfico 11.Rede Neural LVQ (caso A, 1 piora permitida). Taxa de Erro X Épocas

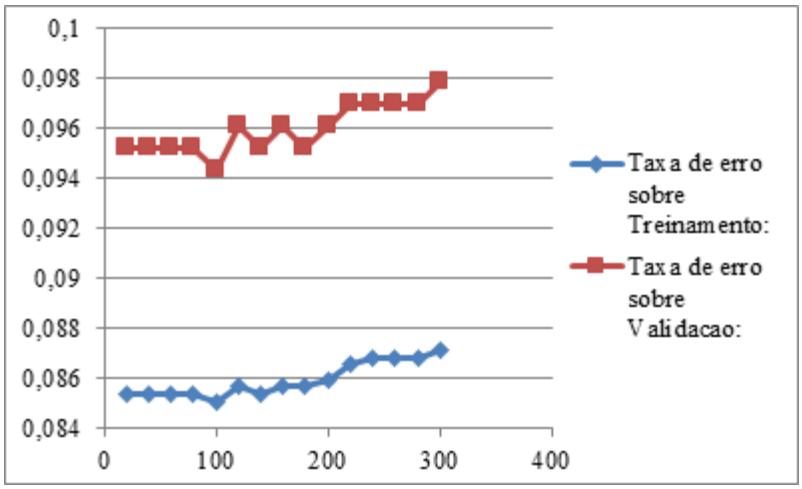

Gráfico 12. Rede Neural LVQ (caso D, 3 pioras permitidas). Taxa de ErroX Épocas

Com esses gráficos pode-se notar que a única coisa que o número de a limitação de pioras no valor um proporcionou, é o processamento de mais épocas que só tenderiam a piorar o treinamento.

No entanto, existem casos em que a ocorrência de apenas uma piora não garante que a rede está para entrar em uma faixa que só tende a piorar o treinamento, podendo assim ser somente um surto momentâneo em que logo em seguida o treinamento irá continuar a rodar normalmente, melhorando os resultados a cada época, como o esperado. A seguir temos um caso assim, através da representação gráfica dos casos H e I, onde a diferença entre os parâmetros dessas arquiteturas se dá apenas na quantidade de pioras permitidas.

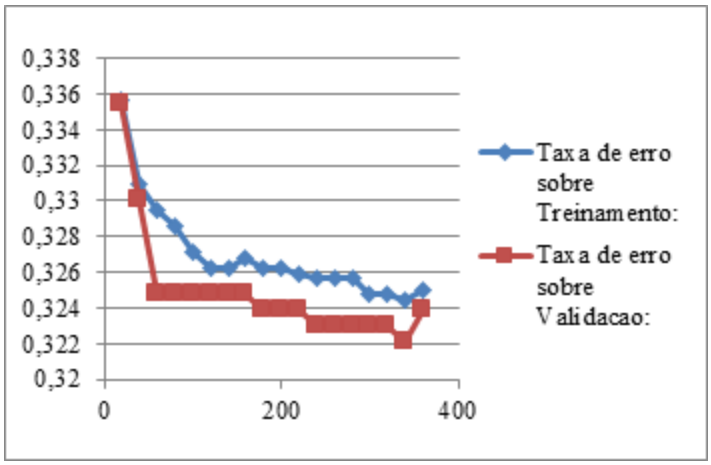

Gráfico 13. Rede Neural LVQ (caso H, 1 piora permitida). Taxa de ErroX Épocas

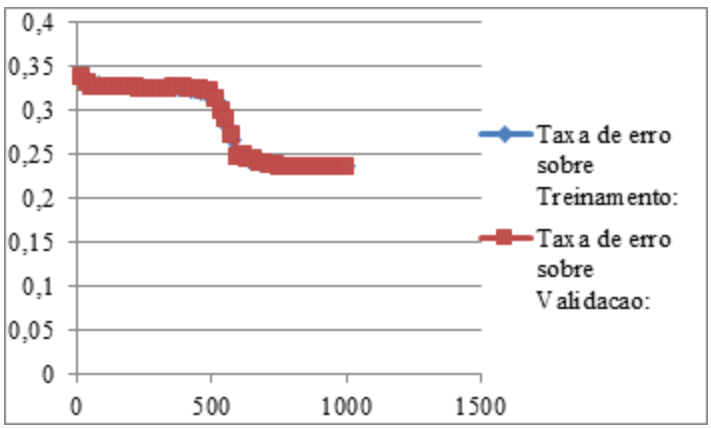

Gráfico 14. Rede Neural LVQ (caso I, 3 pioras permitidas). Taxa de ErroX Épocas

Pode se notar que ele parou durante a época 400, no entanto se considerarmos o gráfico com uma visão um pouco maior, poderemos ver que depois da época 500 irá haver um decaimento bom, que ira melhor bem o aprendizado da rede, e que no caso A acabou sendo desconsiderada essa parte.

b.2.3. Comportamentos interessantes segundo o uso de mais ou menos neurônios de saída.

O uso de mais ou menos neurônios de saída está ligado diretamente com o limite no qual a rede consegue chegar, assim como a taxa de aprendizado, ele está ligado diretamente no desenvolvimento da rede neural na etapa de aprendizagem.

Considerando novamente o caso A, no entanto agora comparado aos casos E e F, podemos verificar o que o aumento de neurônios de saída pode proporcionar no aprendizado da lvq, por menor que seja a diferença entre a quantidade de erros (lembrando que quanto menor a

taxa de erro, mais a rede aprendeu), deve se considerar que houve uma melhora devido ao aumento de vetores de pesos. Assim se observarmos bem, conforme foi se aumentando esse número maior é o limite que a rede consegue chegar.

Outro fator interessante que pode se observar é que o aumento de neurônios não tem influência somente no limite que a rede consegue chegar, mas também na velocidade que ela aprende. Podemos verificar que no caso F o neurônio só passou 78 vezes pelo treinamento e conseguiu atingir uma melhora considerável quando comparado ao caso A que possui somente um neurônio por classe.

No entanto quando levamos em conta o caso E e o caso F, podemos verificar que a diferença no limite que eles conseguem chegar não muda muito, assim devemos considerar se caso o aumento de neurônios e consequentemente a adição no gasto de processamento causado por essa adição de pesos, vale a pena para uma quantidade de ganho no treinamento.

Assim para entender melhor a diferença que a quantidade de neurônios causa no processamento podemos considerar os seguintes gráficos:

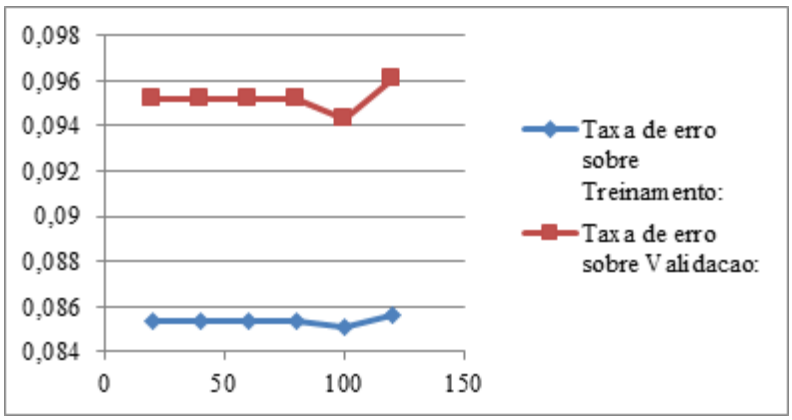

Gráfico 15. Rede Neural LVQ (caso A, 1 neurônio por classe). Taxa de ErroX Épocas

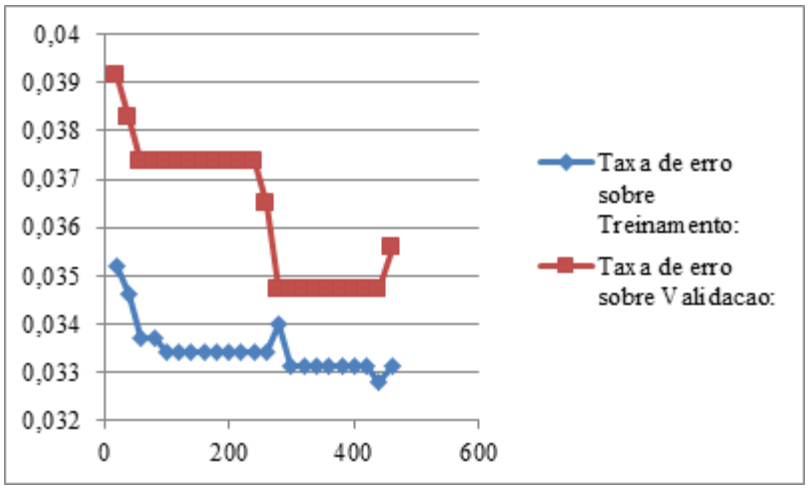

Gráfico 16. Rede Neural LVQ (caso E, 5 neurônios por classe). Taxa de ErroX Épocas

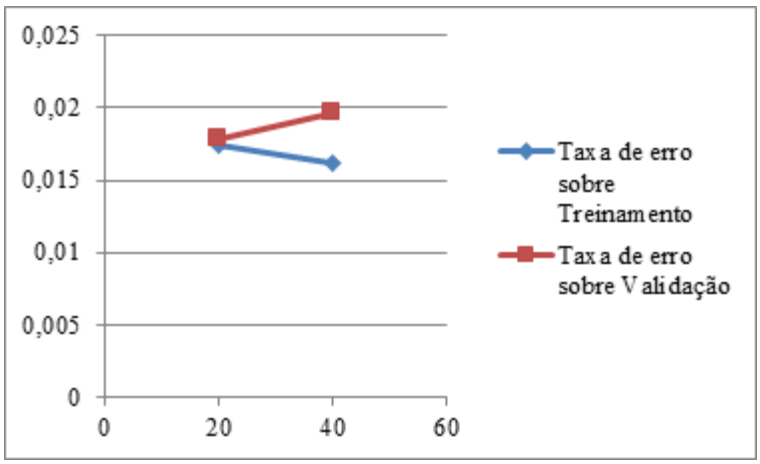

Gráfico 17. Rede Neural LVQ (caso F, 10 neurônios por classe). Taxa de ErroX Épocas

Nesses gráficos podemos observar claramente que logo de inicio há muita diferença boa na taxa de erro conforme a quantidade de neurônios aumenta, é interessante também juntar com as informações referentes à quantidade de épocas necessárias para se atingir uma determinada taxa de erro.

Outro fator que esta altamente relacionada às melhorias causadas pelo aumento de neurônios, é o tipo de inicialização desses neurônios. Em alguns casos o aumento da quantidade de neurônios chega até a degradar o aprendizado, pois a lvq acaba encontra dificuldade para trabalhar com muitos neurônios e acaba se confundindo.

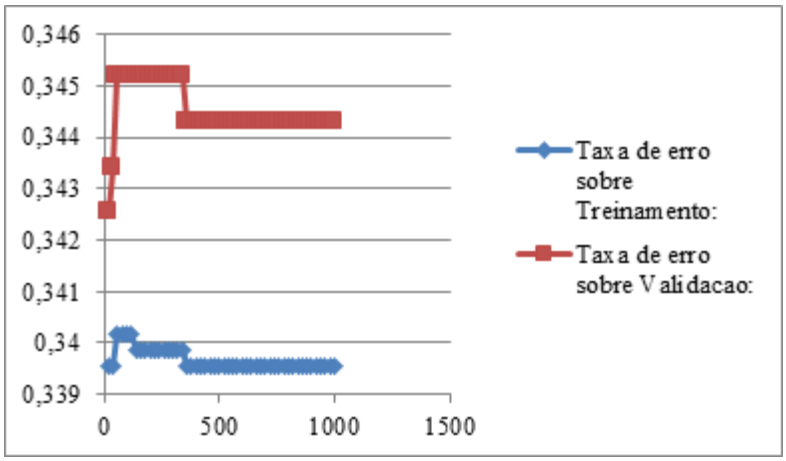

Gráfico 18. Rede Neural LVQ (caso J, 5 neurônios por classe). Taxa de ErroX Épocas

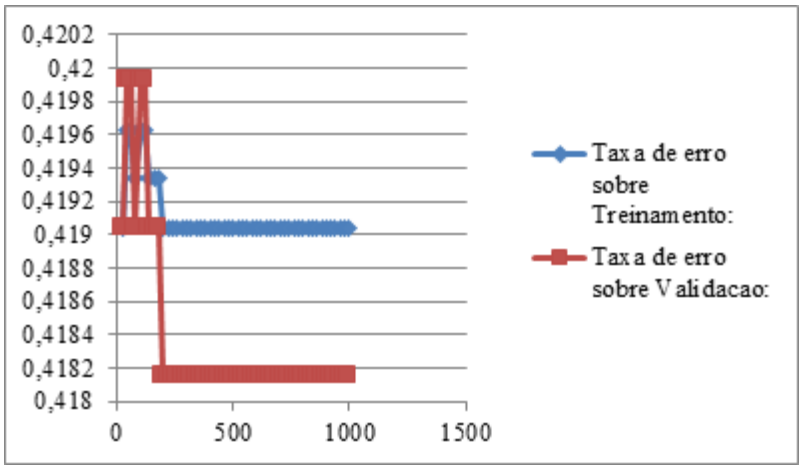

Gráfico 19. Rede Neural LVQ (caso L, 10 neurônios por classe). Taxa de ErroX Épocas

Verificando segundo as taxas de erro que realmente, com mais neurônios, para esse caso o aprendizado tende a ser pior, verificando que logo de inicio ele já começa com uma classificação inferior para maiores quantidades de pesos.

b.2.4. Comportamentos interessantes segundo o tipo de inicialização dos neurônios de saída.

Nesse caso podemos partir de uma abordagem um pouco diferente em relação aos outros estudos. Podemos comparar os melhores casos de teste para cada tipo de inicialização diferente. O que por acaso acabou sendo a mesma arquitetura para os casos, mudando somente o tipo de inicialização.

Logo de inicio podemos começar pelo fato de que a inicialização por primeira entrada acaba sendo superiores as outras em questão de quantidade de aprendizado por número de épocas rodadas.

Podemos considerar que isso se deve ao fato dessa inicialização literalmente roubar dados do conjunto de treinamento, onde ele vasculha as primeiras entradas nos dados de treinamento, levando em conta as diferentes classes, e assume como estes os neurônios de saída, ou seja, se caso tivéssemos 10 neurônios de saída (uma para cada classe definida), eles iriam assumir a forma dos 10 primeiro dados com classes diferentes. Isso acaba por dar a arquitetura com esse tipo de inicialização uma vantagem enorme quando comparada a inicialização com pesos zerados ou com valores aleatórios.

Assim usaremos os casos G, K e M para comparação, onde estes provaram serem as arquiteturas com os melhores valores quando comparado à média sobre a taxa de erros sobre os testes, levando em consideração os testes feitos (vide anexo A: tabela de sintonização LVQ).

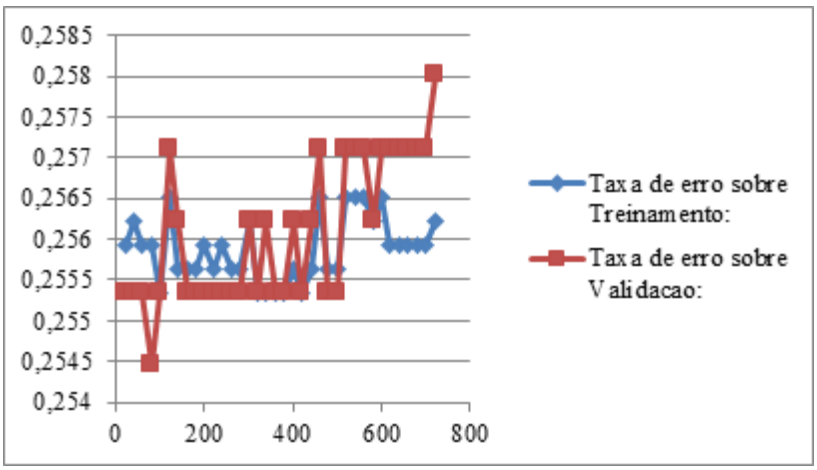

Gráfico 20. Rede Neural LVQ (caso M, inicialização aleatória). Taxa de ErroX Épocas

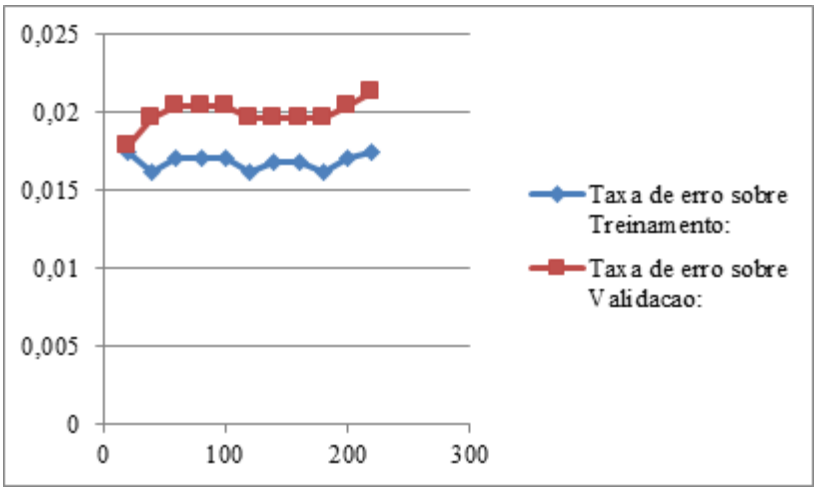

Gráfico 21. Rede Neural LVQ (caso G, inicialização primeira entrada). Taxa de ErroX Épocas

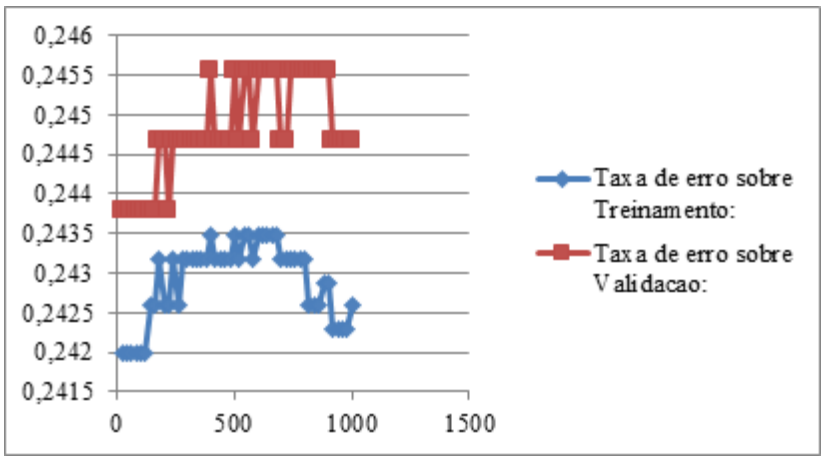

Gráfico 22. Rede Neural LVQ (caso K, inicialização nulo). Taxa de ErroX Épocas

Levando em conta os gráficos acima podemos fazer duas considerações, a quantidade de erro para a inicialização via primeira entrada, acaba sendo muito melhor que as outras duas. E outro ponto que podemos verificar é que por mais que a taxa de erro da inicialização com zero e da inicialização aleatória sejam muito parecidas, verifica que a inicialização por zero se especializa um pouco mais que a aleatória.

Um fato importante que deve ser levado em conta, e que talvez explique os comportamentos muito parecidos para as inicializações de pesos aleatórias e para a inicialização de pesos zerada, é que a inicialização aleatória de dados tem um intervalo muito baixo, podendo ter valores iniciais somente de -1 até 1. Levando em conta que para esse teste se usou os dados brutos, sem qualquer normalização, e que o intervalo de valores para esses dados podem ir de 0 até 16, isso torna o estado inicial das duas redes muito parecida.

5. Avaliação dos classificadores

a. LVQ

Na etapa de classificação tentamos tomar medidas adequadas para tentar verificar o real desempenho dela, assim através de técnicas de *holdout,* executamos em média dez vez cada sintonização e começamos a analisar as informações geradas através desse processo. Todo essa etapa de análise é feita através de medidas de resumo, que tem como o objetivo sumarizar as muitas informações que foram geradas.

Basicamente dentro do estudo de cada arquitetura, nos contemos a análises que levaram em conta medidas, como: matriz de confusão com as médias de acertos, matriz de confusão com o desvio padrão, média da quantidade época alcançada por cada sintonização e média da taxa de erro sobre dados de teste, além de diversos estudos com logs que geramos, onde continham dados como parâmetros da rede, taxa de acerto sobre validação, treinamento, matrizes de confusão, etc. Dentre essas informações muitas foram utilizadas durante o relatório, e é interessante verificar como fizemos essas análises.

Assim como exemplo temos duas matrizes de confusão para ilustrar como avaliar algumas informações sobre as arquitetura LVQs. Na primeira tabela temos a média dos acertos e na segunda o desvio padrão dos acertos.

	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
0.0	111.4	0.0	0.0	0.0	0.4	0.0	0.2	0.0	0.0	0.0
1.0	0.3	90.1	6.2	1.9	3.5	1.1	3.2	3.9	0.0	1.8
2.0	1.3	1.2	108.5	0.0	0.0	0.0	0.1	1.6	0.0	0.3
3.0	2.3	4.0	11.1	64.6	0.0	9.4	0.9	10.4	0.0	10.3
4.0	7.4	12.1	0.4	0.0	62.7	4.0	10.8	12.4	0.0	3.2
5.0	0.4	0.2	0.6	0.1	0.8	105.6	1.2	0.0	0.0	4.1
6.0	0.3	1.1	0.0	0.0	0.3	0.1	110.2	0.0	0.0	0.0
7.0	0.0	0.5	0.2	0.0	0.4	0.5	0.0	110.3	0.0	0.1
8.0	7.5	24.1	16.0	9.1	2.8	20.9	17.3	9.7	0.0	4.6

9.0	**19.6**	**7.9**	**1.9**	**16.2**	**5.2**	**12.8**	**0.8**	**8.2**	**0.0**	**39.4**

Matriz de Confusão com as médias de acerto

	0.0	**1.0**	**2.0**	**3.0**	**4.0**	**5.0**	**6.0**	**7.0**	**8.0**	**9.0**
0.0	**1.32**	**0.0**	**0.0**	**0.0**	**1.29**	**0.0**	**0.63**	**0.0**	**0.0**	**0.0**
1.0	**0.94**	**32.15**	**6.76**	**6.0**	**8.54**	**2.54**	**2.78**	**6.13**	**0.0**	**3.01**
2.0	**1.01**	**1.16**	**2.87**	**0.0**	**0.0**	**0.0**	**0.31**	**1.51**	**0.0**	**0.94**
3.0	**2.72**	**5.05**	**11.23**	**53.36**	**0.0**	**10.41**	**0.92**	**9.65**	**0.0**	**16.2**
4.0	**8.79**	**11.87**	**0.86**	**0.0**	**50.99**	**4.76**	**12.98**	**9.93**	**0.0**	**4.75**
5.0	**0.47**	**0.63**	**1.07**	**0.31**	**1.87**	**5.27**	**1.02**	**0.0**	**0.0**	**5.72**
6.0	**0.48**	**0.90**	**0.0**	**0.0**	**0.87**	**0.31**	**1.47**	**0.0**	**0.0**	**0.0**
7.0	**0.0**	**0.7**	**0.42**	**0.0**	**0.86**	**0.70**	**0.0**	**1.01**	**0.0**	**0.31**
8.0	**4.79**	**10.80**	**5.03**	**9.01**	**3.20**	**4.98**	**4.79**	**4.06**	**0.0**	**6.71**
9.0	**14.11**	**7.27**	**1.93**	**16.11**	**5.99**	**7.41**	**0.94**	**3.95**	**0.0**	**50.92**

Matriz de Confusão com o desvio padrão de acerto

Então com algumas analogias simples, se começarmos a estudar essas tabelas podemos verificar que em maioria dos campos da matriz de média existe algum valor diferente de zero,

assim verificamos que essa arquitetura acaba errado nas diversas classificações, mas se olharmos a diagonal principal, ela tende a acertar bem, não chega a ser um treino perfeito, mas parece ser bom. Já com outra matriz conseguimos verificar o desvio que existe nos acertos, assim verificando se existem casos em que acabamos por errar muito e em outro teste acertar muito.

MLP

O melhor desempenho médio que encontramos para uma rede MLP foi 1,9% de erro no conjunto de teste. Esse resultado foi conseguido pela seguinte estrutura:

- 15 neurônios na camada escondida
- 10 neurônios na camada de saída
- 0,001 como taxa de aprendizado fixa
- dados de entrada normalizados pelos princípios min-max

Vale observar que os resultados finais calculados são a média dos desempenhos de 10 MLPs com a estrutura dada sobre 10 combinações diferentes de holdouts.

	0	1	2	3	4	5	6	7	8	9
0	108.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	107.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	110.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	109.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	108.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	109.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	106.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	107.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	102.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	101.0

Matriz de confusão média

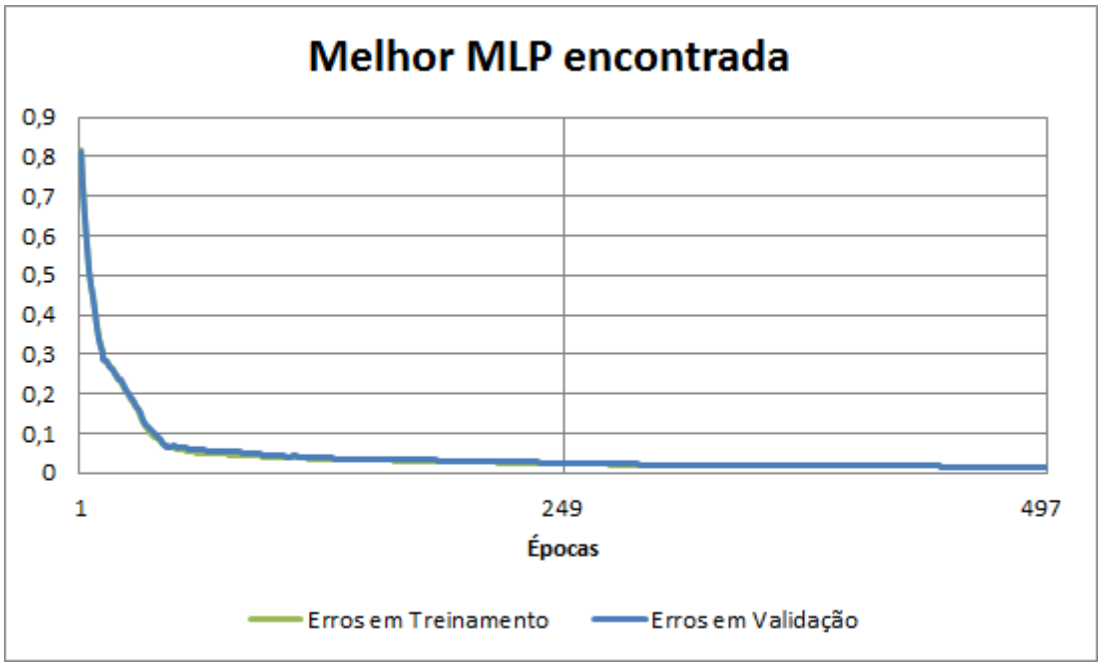

Gráfico 23.

O melhor desempenho encontrado nessa rede foi alcançado depois de 5481 épocas. No entanto, como o ganho foi muito pequena a partir da iteração de número 500, ele foi omitido do gráfico.

6. Bibliografia

[1]http://uhra.herts.ac.uk/bitstream/handle/2299/1364/?sequence=1 ← relaxa q eu sei q tem que ajeitar isso

[2] I.A Basheer, M. Hajmeer. "Artificial neural networks: fundamentals, computing, design, and application". (2000) Journal of Microbiological Methods. Acessado em: 18/05/2015 <http://ethosun2.unige.ch/etho5.10/pdf/basheer.hajmeer.2000.fundamentals.design.and.application.of.neural.networks.review.pdf>

[3] <http://www.cs.cmu.edu/~schneide/tut5/node42.html> Acessado em: 18/05/2015

[4] ROJAS, R. "Neural Networks: A Systematic Introduction". (1996) University of Hale