Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e Computação

EA072 – Turma A (2s2017)

EFC 1

Lucas Rath Maia 117751
Thiago Bruschi Martins 120212

Indice

1. Questão 1

Obtenha um modelo de classificação linear, sendo que os parâmetros do modelo linear devem compor uma matriz de dimensão 785 × 10 e devem ser obtidos de forma fechada, a partir de uma única expressão algébrica. Deve-se buscar um bom parâmetro de regularização, o qual tem que ser maior do que zero, pois a matriz de dados de entrada não tem posto completo. Para tanto, tomar parte dos dados de treinamento como validação para poder implementar esta busca pelo melhor parâmetro de regularização. Uma vez encontrado um valor adequado para o parâmetro de regularização, usar todas as 60.000 amostras de treinamento para sintetizar o classificador linear. Apresentar o desempenho junto aos dados de teste, em termos de taxa média de acerto (considerando todas as classes) e taxa média de acerto por classe. Esses mesmos índices de desempenho devem ser empregados nas outras duas partes da atividade.

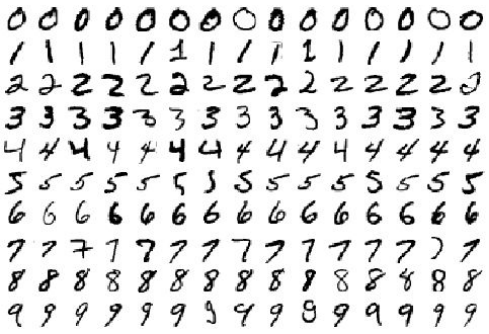


Figura 1: Exemplos de imagens do conjunto de dados MNIST

1.1. Parte 1

Para a primeira parte desta atividade, elaborou-se um modelo de rede neural de classificação linear. Foi definido primeiramente que cada neuronio da camada intermediária se conectaria somente a um pixel de entrada. A função de ativação do m-ésimo neurônio foi definida pelo seguinte comando no Matlab, como mostra a **Equação 1**:

$$act\_fun = @(x,m)\ x(:,mod(m-1,size(x,2))+1).^{ceil(m/size(x,2))}; \qquad \textbf{(1)}$$

Em notação matemática:

$$\begin{aligned} h_m(X) &= 1 && para\ m = 0 \\ h_m(X) &= X^{round(m/784)}_{mod(m,784)} && \forall\ m \in \mathbb{N}^* \end{aligned} \qquad \textbf{(2)}$$

Simplificando a notação, tem-se que para os primeiros 2352 neurônios, a função de ativação pode ser dada por:

$$\begin{aligned} h_m(X) &= 1 && \text{para} && m = 0 \\ h_m(X) &= X_m && \text{para} && 1 \le m \le 784 \\ h_m(X) &= X^2_{m-784} && \text{para} && 785 \le m \le 1568 \\ h_m(X) &= X^3_{m-1568} && \text{para} && 1569 \le m \le 2352 \end{aligned} \qquad \textbf{(3)}$$

Deste modo, se houver ao menos 785 neurônios, todos os pixels de entrada da imagem poderão ter influência no resultado da classificação. No entanto, como especificado no enunciado, foram usados

somente 785 neurônios para esta primeira parte do problema. Ao final da seção, por curiosidade, comparamos a eficiência do classificador linear com o uso de mais neurônios.

Após a definição das funções de ativação, executou-se o treinamento da rede neural linear segundo o **Pseudocódigo 1**, usando somente as 60 mil amostras de treino disponíveis, representadas por X e S. Posteriormente, as 10 mil amostras de teste restantes, foram usadas apenas para sintetizar a eficiência do classificador linear treinado.

Pseudocódigo 1: Treinamento de rede neural linear

função $w =$ **treinar_rede_linear** (X, S)
  Defina aleatoriamente 80% *das amostras como* X_{treino} *e* S_{treino}
  Defina as 20% *restantes como* $X_{validação}$ *e* $S_{validação}$
  Calcule H_{treino} *e* $H_{validação}$
  Para k variando de 1 *até K classes* :
    Inicialize C_k *aleatoriamente maior que zero e pequeno*
    Enquanto critério de parada não for atendido, faça :
      $w(C_k) = (H_{treino}{}^T * H_{treino} + I * C_k)^{-1} * H_{treino}{}^T * S_{treino,k}$
      $MSE(C_k) = \|H_{validação} * w - S_{validação}\|^2 / N$
      Calcule $\nabla MSE(C_k)$
      Defina o passo α
      $C_k = C_k - \alpha * \nabla MSE(C_k)$
    Concatene H_{treino} *e* $H_{validação}$ *e atribua a H*
    $w_k = (H^T * H + I * C_k)^{-1} * H^T * S_k$

A tarefa de treinar a rede neural deste problema consiste em determinar o vetor de pesos sinápticos w_k para cada uma das classes, o qual é um problema de otimização linear. No entanto, a matriz H não possui posto completo, nos levando a incluir um coeficiente de regularização C_k. Assim, foi realizada uma busca unidimensional do coeficiente de regularização para cada uma das 10 classes pelo método de otimização por gradiente. Para cada classe, o valor ótimo de C_k é aquele que minimiza o erro junto ao conjunto de dados de validação, ou seja, minimiza a função *Mean Square Error* (*MSE)*. Este problema de otimização foi executado através do comando *fminunc* do Matlab, o qual calcula a aproximação do gradiente e da matriz Hessiana da função pelo método de Quasi-Newton. No entanto, outras funções também poderiam ter sido usadas para fins de otimização no treinamento da rede neural, como as funções *fminsearch* e *lsqnonlin*. Na **Figura 2**, pode-se ver que o valor da função Mean Square Error (MSE), para um determinado valor de C_k, convergiu e ficou preso em um mínimo local. Para tanto, atingiu-se o critério de convergência e prosseguiu-se então para o próximo valor de C_k. Como uma possível melhoria para o algoritmo, o valor de C_k poderia ter sido encontrado através de algoritmos genéticos de busca, os quais possivelmente encontrariam o mínimo global para um espaço de design determinado.

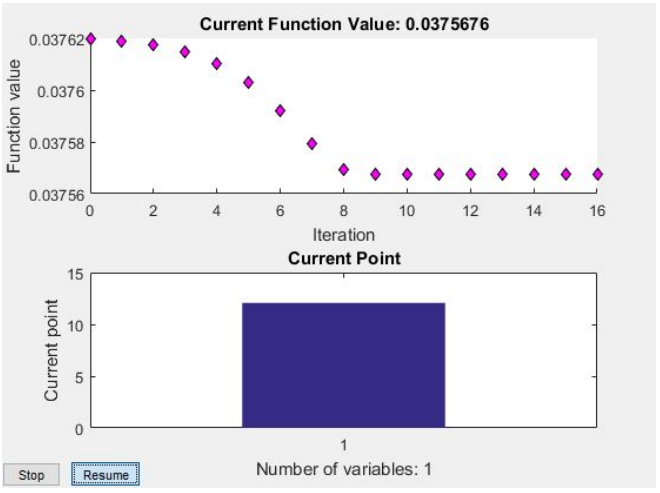


Figura 2: Otimização da função MSE por gradiente do coeficiente de regularização usando a função *fminunc* do Matlab

Ao final do processo de otimização, foram encontrados os seguintes valores de C_k para cada uma das 10 classes, como mostrado na **Figura 3**. No entanto, para cada execução do programa, valores diferentes de C_k foram obtidos para cada classe. Isto confirma a hipótese inicial de que a função de otimização por gradiente ficou presa em um mínimo local da função objetivo *MSE* e não conseguiu convergir para um mínimo global.

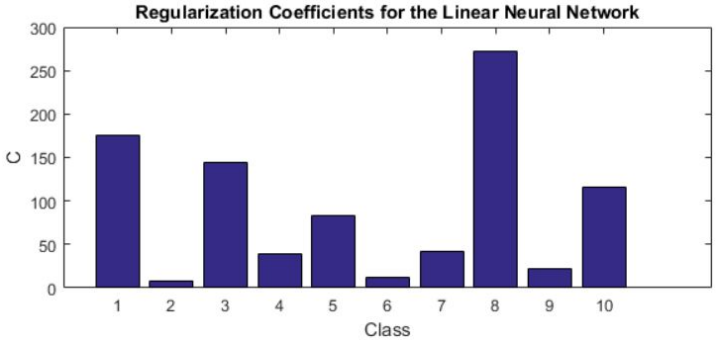


Figura 3: Coeficientes de regularização para a rede neural linear

Outro ponto importante de se notar na **Figura 3**, é que foram obtidos valores muito diferentes para cada coeficiente de regularização. Em resumo, este coeficiente multiplica a norma euclidiana do vetor de pesos sinápticos w e soma-se a função *MSE* para formar a função objetivo de minimização,

$$w_k^* = arg\ \ min_{\,w_k}\ \ MSE(w_k)\ +\ C_k * \|w_k\|^2 \qquad \textbf{(4)}$$

de tal modo que ele "penaliza" altos módulos de w e assim elimina pesos desnecessários que estão provavelmente só ajustando a ruídos. No caso deste trabalho os valores de C_k foram determinados de tal modo que o vetor w_k minimiza o MSE junto aos dados de validação. Isto nos leva a concluir que classes de números muito complexas e que possuam dados muito ruidosos (localização dos pixels ativos no número varia muito), possuirão altos valores de C_k. Para tanto, podemos constatar preliminarmente, que os números 2, 6 e 9 foram os que menos apresentaram complexidade,

mantendo melhor, dentro do possível, o padrão da região de pixels ativos do número. Já os números 1, 3 e 8 podem ser considerados os mais complexos.

Uma vez encontrado um valor adequado para o parâmetro de regularização, foram usadas todas as 60.000 amostras de treinamento para sintetizar o classificador linear. Ao final do treinamento, foi obtido o vetor de peso sináptico para as 10 classes, resultado em uma matriz 785x10. Felizmente, para cada classe, podemos formar uma matriz 28x28 de pesos sinápticos, que estão ligados um a um a cada um dos pixels 28x28 pixels de entrada da rede neural. Essa matriz, está representada graficamente por um gráfico de calor, como mostrado na **Figura 4**.

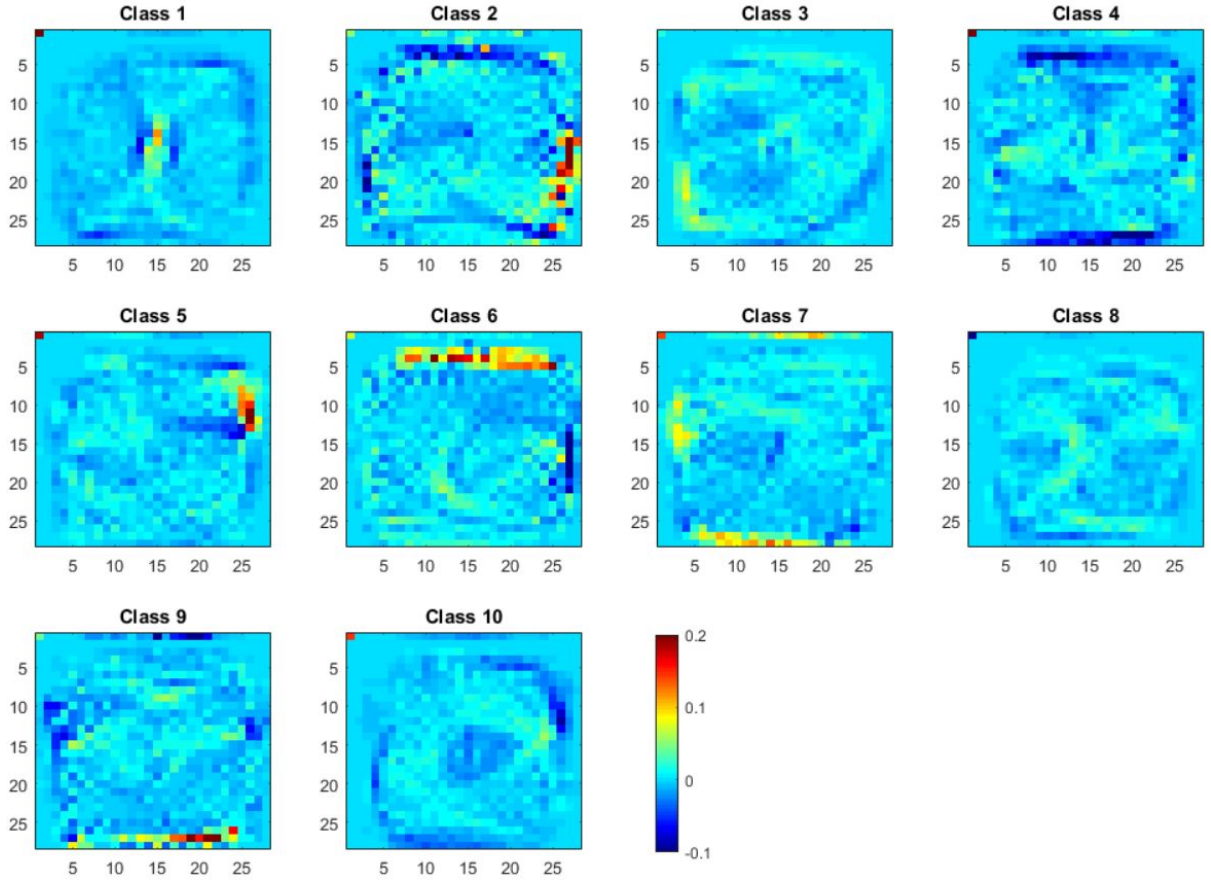


Figura 4: Pesos sinápticos para cada neurônio de entrada

Observa-se que a classe no número zero (Class 10) por exemplo, possui pesos negativos para os pixels que estão localizados no centro da imagem e pesos maiores para uma região circular que lembra exatamente o número zero. Similarmente, com um pouco de esforço, é possível reconhecer o número de cada classe no mapa de calor pela região delimitada pelos pesos sinápticos maiores. Esta observação é extremamente importante, e nos leva a concluir que o vetor dos pesos sinápticos da camada final tende a ter aproximadamente a direção média dos vetores dados de entrada, dos quais possuem saída ativa, ou seja, dos quais representam a classe a ser identificada.

Para efeitos de comparação da eficiência dos classificadores, considerou-se no fim, somente uma saída por classe, de tal modo que a classe indicada é aquela associada à saída de maior valor

numérico. A medida denominada Classification Error Rate (*CER*) contabiliza os erros cometidos em cada classe e divide pelo total de amostras de cada classe, para obter uma taxa de erro por classe.

Por fim, o mesmo algoritmo foi executado para 4 estruturas diferentes da camada de neurônios intermediária, resultado em diferentes erros associados à classificação. Os resultados estão mostrados na **Tabela 1**.

Tabela 1: Relação entre o número de neurônios da camada intermediária e o índice *CER*

Hidden Layer Structure	CER
[784]	0.14
[1568]	0.11
[2352]	0.10
[3136]	0.10

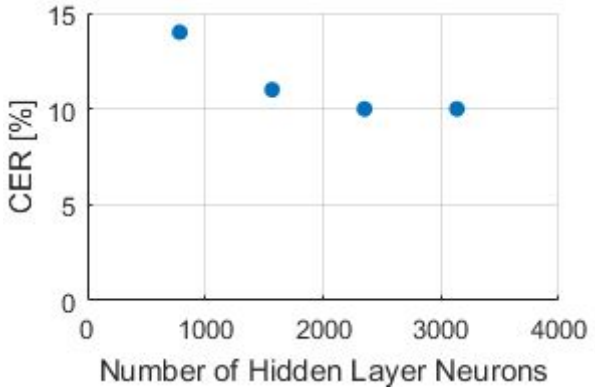


Figura 4: Relação entre o número de neurônios da camada intermediária e o erro *CER* associado aos dados de teste

Pôde ser constatado que quanto maior o número de neurônios utilizados, menor o erro de classificação apresentado pela rede neural, junto ao dados de teste. Isto se deve ao fato de que a rede possuiu cada vez mais flexibilidade para se ajustar aos dados usados no treinamento. Contudo, o aumento excessivo do número de neurônios poderia ter causado *overfitting*, o que levaria a um aumento do *CER*, comparada a uma configuração com menos neurônios.

1.2. Parte 2

A tarefa referente à segunda parte do exercício deveria ser realizada utilizando máquinas de aprendizado extremo (Extreme Learning Machines). Aqui, o maior desafio, consiste em determinar o número de neurônios da camada intermediária e as características da distribuição aleatória dos pesos sinápticos da camada de entrada. Para tais tarefas, recorremos a 2 papers renomados na área.

O primeiro paper refere-se a adição incremental, um a um, de neurônios da camada intermediária de tal modo que o vetor de pesos sinápticos da camada de saída são calculados muito mais rapidamente em comparação a inicialização de uma nova ELM com um neurônio a mais. Esta máquina levou o nome de *IR-ELM*, *Incremental Regularized ELM* **[ZXU14]**. Em comparação com a ELM original, a complexidade de cálculo baixou de $O(m\,x\,N^2)$ para $O(m\,x\,N)$, quando necessário elevar unitariamente para um número total de m de neurônios da camada intermediária e para N amostras de treino disponíveis. No paper, são mostradas que as possíveis aplicações da melhoria sugerida, estão entre:

- Para um determinado erro máximo desejado, determine o número mínimo de neurônios da camada intermediária necessários para se obter um erro abaixo ao desejado.
- Para um conjunto de dados de treino e validação, determine o menor erro possível que a rede neural pode atingir; o que, por consequência, exige definir o número de neurônios necessários da camada intermediária

Estes dois propósitos se encaixam perfeitamente na tarefa dessa questão, uma vez que queremos estudar como o erro e o tempo de processamento variam com o aumento de neurônios utilizados na camada intermediária para o problema MNIST. Tal estudo, com uma ELM original, se tornaria quase proibitivo, pelo tempo de processamento total necessário para obter resultados tão detalhados como os que serão apresentados aqui.

No entanto, fizemos uma pequena melhoria no algoritmo de *M. Yao*, já que não foram considerados a otimização e nem diferentes coeficientes de regularização para diferentes neurônios de saída. Por fim, o pseudo-código final pode ser visto no **Pseudocódigo 2**. Assim, deve-se ter uma função para treinar a rede ELM com um número m inicial de neurônios e uma outra função para incrementar unitariamente os neurônios da camada intermediária:

Pseudocódigo 2: Treinamento e incremento de uma IR-ELM melhorada

função $w =$ **treinar_rede_IR_ELM** (X, S)
Defina aleatoriamente 80% *das amostras como* X_{treino} *e* S_{treino}
Defina as 20% *restantes como* $X_{validação}$ *e* $S_{validação}$
Defina aleatoriamente todos os pesos de entrada w para os m neurônios da camada intermediária
Calcule H_{treino} *e* $H_{validação}$
Para k variando de 1 *até K classes* :
Inicialize C_k *aleatoriamente maior que zero e pequeno*
Enquanto critério de parada não for atendido, faça :

$$w(C_k) = (H_{treino}{}^T * H_{treino} + I * C_k)^{-1} * H_{treino}{}^T * S_{treino,k}$$

$$MSE\ (C_k) = \|H_{validação} * w - S_{validação}\|^2 / N$$
$$Calcule\ \nabla MSE(C_k)$$
$$Defina\ o\ passo\ \alpha$$
$$C_k = C_k - \alpha * \nabla MSE(C_k)$$
$$Concatene\ H_{treino}\ e\ H_{validação}\ e\ atribua\ a\ H$$
$$w_k = D_k * S_k = (H^T * H + I * C_k)^{-1} * H^T * S_k$$

$função\ \ w =$ **crescer_rede_IR_ELM** (X, S)

$$m = m + 1$$
$$Adicione\ um\ novo\ neurônio\ e\ atribua\ aleatoriamente\ pesos\ sinápticos\ de\ entrada\ a\ ele$$
$$H_m = [H_{m-1},\ v_m],\ tal\ que\ v_m\ é\ o\ vetor\ de\ valores\ de\ saída\ do\ novo\ neurônio\ adicionado$$
$$Para\ k\ variando\ de\ 1\ até\ K\ classes:$$
$$M_{k,m} = v_m^T * (I - H_{m-1} * D_{k,m-1}) / (v_m^T * (I - H_{m-1} * D_{k,m-1}) * v_m + C_k)$$
$$L_{k,m} = D_{k,m-1} * (I - v_m * M_{k,m})$$
$$w_k = D_{k,m} * S_k = [L_{k,m}\ M_{k,m}]^T * S_k$$

Vale ressaltar, de que os pesos sinápticos da camada de entrada foram definidos aleatoriamente. No caso deste trabalho, decidimos por obter cada peso por uma distribuição de *Monte-Carlo* no intervalo [-1,1].

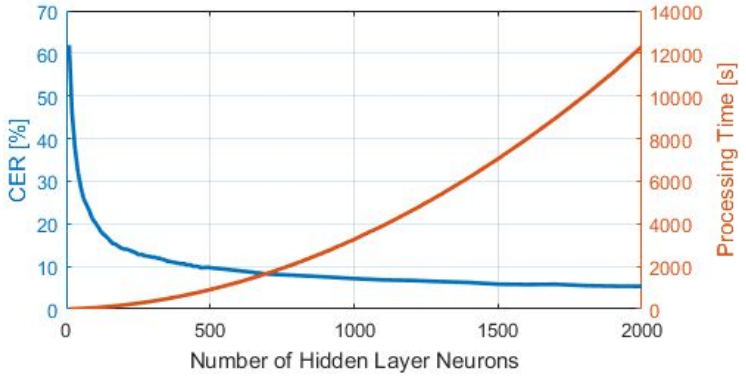


Figura 5: Erro e tempo de execução total para diferentes números de neurônios da camada intermediária para a rede *IR-ELM*

No entanto, sabemos que a atribuição completamente aleatória dos pesos sinápticos de entrada não é uma boa estratégia, já que pode acontecer de ser atribuído um vetor de pesos quase ortogonal aos vetores dos dados de entrada que ativam uma determinada classe. Este neurônio terá então contribuição quase nula para a diminuição do erro, pois ele pouco será ativado para esta classe. Assim, recorremos à literatura, mais especificamente à máquina *CIW-ELM, Computed Input Weights ELM* **[TAP14]**. Neste trabalho, o autor demonstra que os pesos sinápticos não devem ser atribuídos de forma aleatória, mas através de uma combinação aleatória linear dos vetores dado de entrada. Deste modo, demonstrou-se que a rede *CIW-ELM* precisou de muito menos neurônios para atingir a convergência quanto ao erro mínimo. O **Pseudocódigo 3**, mostra de forma sintética, como é a

atribuição de pesos sinápticos para a camada de entrada da rede *CIW-ELM* e que foi usada nesta questão.

Pseudocódigo 3: Atribuição dos pesos sinápticos de entrada para rede *CIW-ELM*

$função \quad w =$ **gera_pesos_sinapticos** (X)
$Gere\ uma\ matriz\ R_{m\,x\,N}\ contendo\ somente\ \{-1, 1\}\ aleatoriamente$
$w = (R * X)^T$
$w_a = w_a / |w_a|,\ para\ cada\ a \in [1, m],\ tal\ que\ a$ é *um índice da coluna*

O resultado final destas etapas, foi de que reunimos os benefícios da rede *IR-ELM*, da rede *CIW-ELM*, e ainda da otimização dos coeficientes de regularização para criar uma *ELM* nova, a *CIW-IR-ELM*, mais rápida computacionalmente e que apresenta menor erro para um menor número de neurônios em comparação com a rede *ELM* tradicional. Deste modo, quando incrementados neurônios, estes também serão inicializados com pesos sinápticos através de uma combinação aleatória e linear dos vetores dados de entrada. As melhorias obtidas nestes experimentos podem ser conferidas nas **Figuras 6 e 7**.

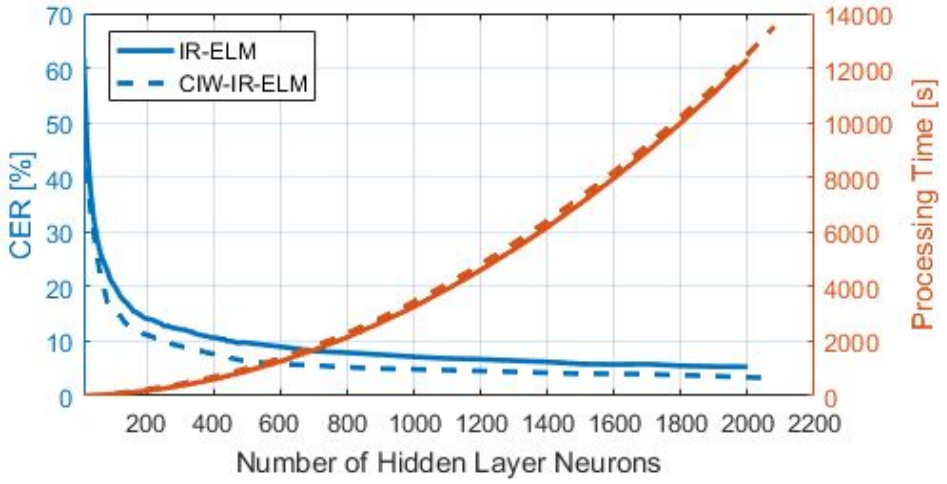


Figura 6: Erro e tempo de execução total para diferentes números de neurônios da camada intermediária

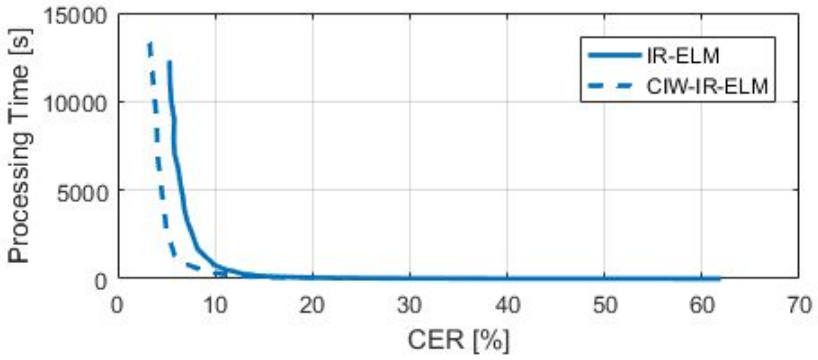


Figura 7: Tempo computacional total necessário para se atingir um determinado erro

Pode-se concluir então, que a rede *CIW-IR-ELM* elaborada por nós nesta questão, apresenta menor erro para o mesmo número de neurônios que a *IR-ELM* e também por consequência, precisa de menor tempo computacional para atingir um mesmo erro final desejado.

Salientamos também, de que foi usada uma função de ativação logística para todos os neurônios da camada intermediária. Esta função foi escolhida pois tem imagem [0 1], exatamente o intervalo dos dados de saída esperados para cada classe. Deste modo, os pesos sinápticos de saída serão pequenos e em torno de zero, o que facilitaria a busca pelo coeficiente ideal de regularização. No entando, não foram notadas diferenças significativas de erro e tempo de processamento em comparação com função tangente hiperbólica.

Por fim, como os algorítmos da *IR-ELM* e *CIW-ELM* foram desenvolvidos somente para uma única camada intermediária, resolvemos por manter esse padrão, de forma a poder usa-los na representação deste problema.

1.3. Parte 3

Para a terceira parte desta tarefa, optamos por empregar as redes neurais MLP com somente uma camada intermediária, de modo a poder fazer uma análise comparativa com as duas primeiras partes desta questão. Logo de início, determinar o número adequados de neuronios da camada intermediária para este problema é uma tarefa não muito simples, já que esta decisão depende de diversos fatores como complexidade do mapeamento a ser aproximado, níveis de ruido, processamento computacional disponível, entre outros. Assim, resolvemos testar diferentes números de neuronios da camada intermediária e observar o erro CER final para os dados de validação. Estes dados podem ser vistos na **Figura 8**.

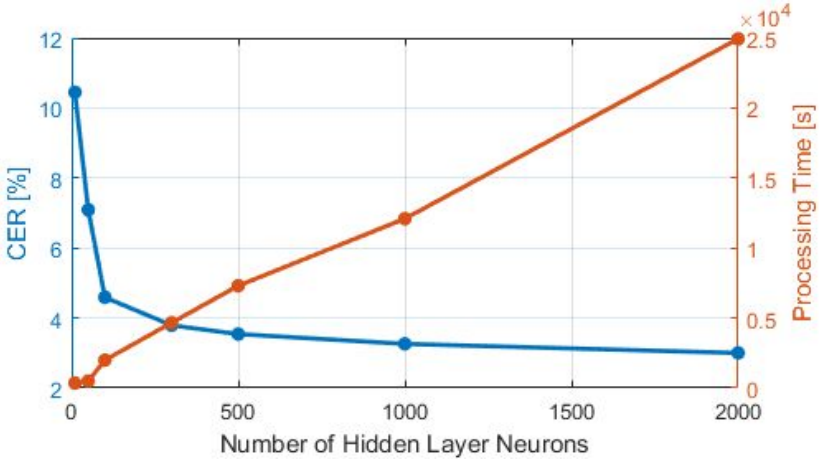


Figure 8: CER e tempo computacional exigido para diferentes numeros de neuronios da camada intermediária

Os resultados obtidos por meio de uma única camada intermediária se mostraram satisfatórios. É notório que com poucos neorônios (<100), já é possível obter resultados razoáveis com erros menores que 10%, tal que o erro de classificação chegou a atingir 3% com 2000 neuronios na camada intermediária. Fica possível identificar, que com o aumento de neurônios, passando dos 2000, o erro continuaria a cair.

Podemos observar também na **Figura 9**, que as 300 épocas escolhidas como limite para treinamento são razoáveis, porém não foi um valor ideal, pois o erro aparentemente ainda continuaria a cair durante o processo de otimização. Não foram escolhidas mais de 300 épocas para treinamento devido ao tempo computacional demandado para treinamento que necessitaria para a execução em nossos computadores. Deste modo, o ideal seria treinar um número de épocas, até que o erro de validação (CER) começasse a subir, caracterizando um *overfitting* do mapeamento. Assim escolheriamos então o treinamento que obtivesse o menor erro CER para validação como rede neural final para o modelo de predição.

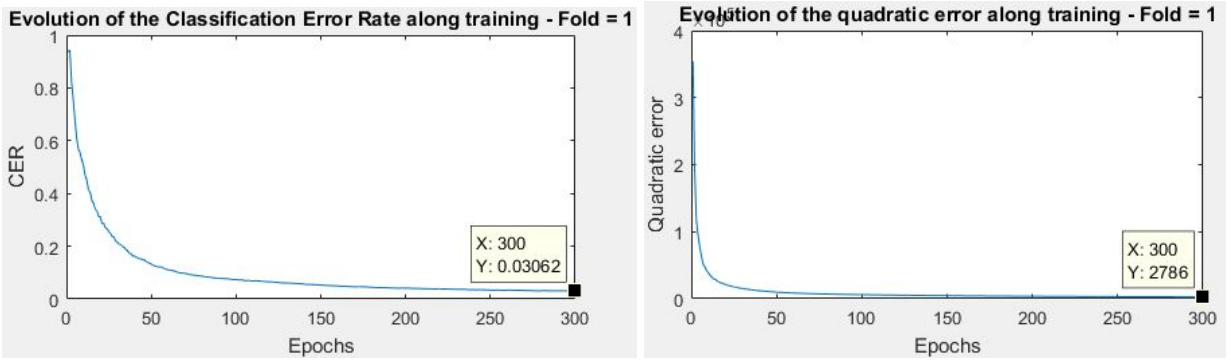


Figure 9: Evolução do CER da validação e MSE dos dados de treinamento para o treinamento da rede neural com 2000 neurônios na camada intermediária

No entanto, embora esta estrutura de rede neural tenha sido usada, resultados ainda melhores poderiam ser facilmente obtidos com o uso de mais camadas intermediárias. Uma boa estratégia para resolver este problema usando redes MLP é proposto em **[MAT17]**. Nesta abordagem, fez-se o uso de *encoders* nas 3 primeiras camadas intermediárias da rede neural. Deste modo, há uma redução da dimensão para um subspaço de variáveis cada vez mais úteis, relevantes e menos redundantes. A ultima camada então, seria responsável então por classificar as variáveis do subspaço entre as diferentes classes. O resultado final, foi um erro CER de 1.4% para este problema MNIST. Nosso grupo resolveu então por não reproduzir este tipo de solução, uma vez que demandaria um processamento computacional muito elevado.

1.4. Consolidação dos Resultados

Os resultados obtidos nos métodos implementados são apresentados na **Tabela 2**. Como é de se esperar, nossos resultados não são comparáveis com o estado da arte. No entanto, dado a complexidade dos algoritmos e os recursos usados pelo melhor método apresentado **[WAN13]**, ficamos satisfeitos com o resultado obtido: um erro cerca de dez vezes maior que os melhores algoritmos da atualidade. Vale ressaltar que das 50 máquinas de aprendizado que apresentaram o melhor desempenho no problema MNIST, quase todas elas se baseiam em conceitos de *Deep Learning*, apresentando muitas camadas de neuronios intermediárias através de varias etapas de extração de padrões relevantes e úteis.

Tabela 2: Tabela comparativa de resultados dos métodos implementados com o estado da arte

Método	**Erro (%)**
DropConnect	0.21
Multi-column Deep Neural Networks	0.23
Recurrent Convolutional Neural Network	0.31
MLP	3.00
CIW-IR-ELM	3.25
IR-ELM	5.32
Linear	10

Dentre os métodos desenvolvidos por nós, podemos notar uma grande diferença entre o método linear, e os demais. Na rede ELM, temos uma notável melhora (de 5.32 para 3.25) quando os pesos deixaram de ser totalmente aleatórios, através do método Computed Input Weights.

2. Questão 2

(1) Recorra ao paper [Guyon, I.; Elisseeff, A. "An introduction to variable and feature selection", Journal of Machine Learning Research, vol. 3, pp. 1157-1182, 2003] para explicar a diferença entre filtro e wrapper.

Ambas as técnicas de filtragem e *wrapping* tem como objetivo principal selecionar um subconjunto de variáveis, dentre um conjunto grande de variáveis de entrada, tal que este subconjunto tenha poder preditivo suficiente para representar o modelo matemático em questão. Alguns benefícios obtidos pela redução do conjunto de variáveis de entrada estão: melhora do performance preditiva do modelo, trainamento mais rápido e com menor custo computacional e melhor entendimento do processo que gerou os dados disponíveis.

No entanto, filtros e wrappers se distinguam em pontos importantes. No caso dos filtros, a seleção de conjuntos de variáveis é baseada em um pré-processamento independente da escolha do preditor que posteriormente será usado. A maioria dos filtros se baseia na seleção de variáveis por métodos de ranqueamento de acordo com a sua relevância, ou seja, com a sua correlação ou informação mútua com os dados de saída. Esta é uma opção muito usada na literatura devida a sua simplicidade, escalabilidade e boa performance empírica.

Já os *wrappers* utilizam o aprendizado de máquina de interesse como uma caixa preta, para selecionar um subconjunto de variáveis úteis de acordo com o seu poder preditivo representado. Por usar as máquinas de aprendizado como uma *black-box* , *wrappers* podem ser considerados universais e muito simples. No entanto, são muitas vezes criticados devido ao seu uso da "força bruta" por usar quantidades massivas de computação.

Por fim, vale ressaltar a diferença entre construir subconjuntos úteis e relevantes. Os filtros muitas vezes selecionam um subconjunto de variáveis relevantes, mas que no entanto podem ser redundantes e assim não possuem poder preditivo suficiente para representar o modelo. Já os *wrappers* selecionam subconjuntos úteis, podendo excluir muitas variáveis relevantes, mas que são redundantes e não necessárias para a predição do modelo. Assim, filtrar pode ser muitas vezes subótimo para construir um modelo, já que as variáveis podem ser redundantes. Métodos que pontuam variáveis individualmente e independentemente com relação as outras variáveis pecam em determinar a combinação de variáveis que trará a melhor performance.

(2) Explique como funcionam as abordagens forward selection e backward elimination e apresente a razão pela qual elas não garantem encontrar a melhor combinação de entradas para a tarefa. Aponte vantagens e desvantagens da abordagem backward elimination

Na abordagem de *forward selection*, as variáveis do subconjunto final são incorporadas progressivamente. A variável adicionada será aquela que junto ao subconjunto atual, mais decresce o erro quadrático médio (MSE) do modelo preditivo da máquina de aprendizado em questão.

Já na abordagem *backward elimination*, começa-se com o conjunto total de variáveis de entrada. Variáveis são então removidas uma a uma, tal que a removida será aquela a qual possibilita o conjunto restante apresentar o menor erro quadrátivo médio (MSE).

No meio acadêmico, a *forward selection* é usualmente denominada mais eficiênte computacionalmente do que a *backward elimination* na escolha de subconjunto de variáveis. No entanto, este subconjunto é geralmente mais fraco, já que a importancia de variáveis que estão sendo incluidas não são avaliadas em conjunto com variáveis que ainda não estão no subconjunto atual. Para exemplificar melhor este problema, vamos dizer que temos um modelo com 3 variáveis (x1, x2 e x3) de tal modo que queremos reduzir a dimensão do problema. Sabemos que isoladamente x3 é a variável que tem maior poder de predição, no entanto, as variáveis x1 e x2 em conjunto, formam o subconjunto com melhor poder preditivo. No caso da abordagem *forward selection*, primeiramente será selecionada a variável x3 e posteriormente a variável x1 ou x2. No entanto sabemos que este não é o melhor subconjunto. Já na abordagem *backward elimination*, a variável x3 seria eliminada de imediato, embora ela sozinha tenha o melhor poder preditivo. Resultante deste processo seria então o melhor subconjunto. Assim, duas variáveis que podem ser inúteis sozinhas, podem ter um alto poder preditivo quando juntas e estas serão levadas em consideração pela *backward elimination*.

(3) a. Use os programas do diretório [wrapper_sunspot]. Comente acerca da série temporal sunspot: a que seus valores se referem, qual é o período estimado desta série, qual é a periodicidade dos valores medidos, etc.

A *sunspot series* é um banco de dados com registros mensais do número estimado de manchas solares observados da terra entre os anos de 1749 e 2010, como pode ser visto na **Figura 10**. Manchas solares, são fenomenos temporários e periódicos na photosfera do sol, tal que podem ser identificados por áreas escuras com relação ao seu redor. São regiões com temperatura superficial reduzida causada pela concentração de fluxos de campos magnéticos que inibem a convecção de calor na região.

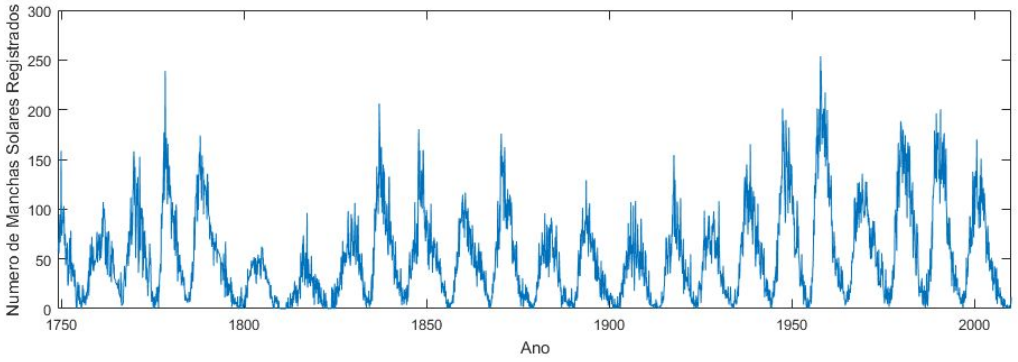


Figure 10: Numero estimado de manchas solares registrados para cada mês entre 1749 e 2010

Como pode ser constatado na figura, parece ocorrer uma certa peridiocidade na aparição de manchas solares. Tal período pode ser identificado através de uma transformada de Fourrier para o domínio da frequência, como pode se visto na **Figura 11**.

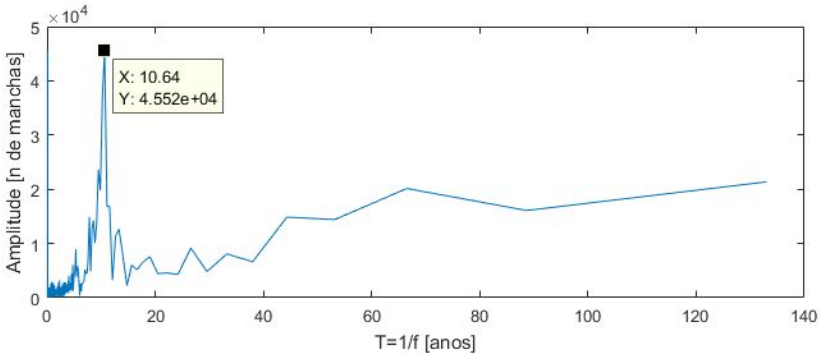


Figura 11: Transformada de Fourrier para o conjunto de dados *Sunspot Series*

Assim, a peridiocidade mais acentuada das manchas solares está em torno de 10.6 anos. Outra peridiocidade em torno de 70 anos também pode ser observada, porém de forma menos acentuada.

(3) c. *Use o programa [lin_filter.m] com argumento 'train' de modo a obter as correlações lineares de Pearson entre as 25 entradas candidatas e a saída (predição a ser realizada). Comente os resultados obtidos.*

Executando o filtro linear, obtivemos as seguintes correlações de Person para cada umas das 25 entradas candidatas, como pode ser visto na **Figura 12**.

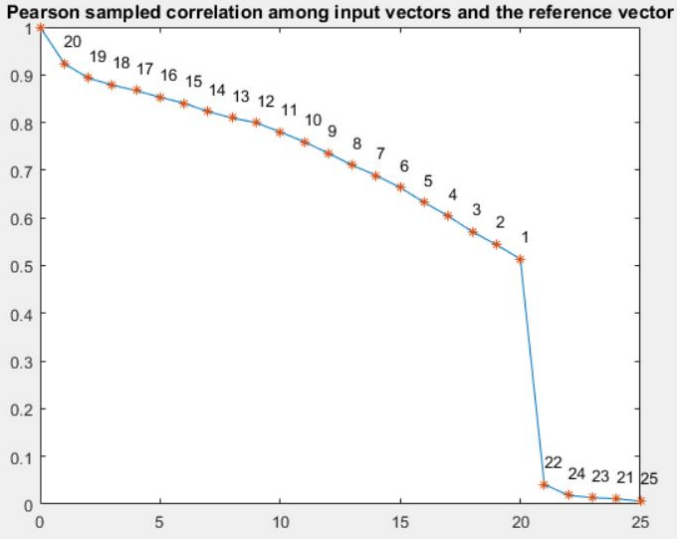


Figura 12: Coeficientes de correlação de Pearson

Como pode ser constatado, as variáveis foram ordenadas independentemente e isoladamente de acordo com seu grau de correlação e covariância com a saída. Ainda, as variáveis mais relevantes são aquelas que possuem o menor atraso temporal. Por ultimo, obviamente, estão as variáveis de prova aleatórias.

Caso adotemos este método de filtragem para escolher um subconjunto de variáveis e reduzir o problema para um subespaço menor, eliminaremos com certeza as 5 variáveis de prova adicionadas. No entanto, como abordado anteriormente, a ordem de relevância estabelecida é feita de modo independente e isolado, de tal modo que possivelmente selecionaremos muitas variáveis redundantes e excluiremos variáveis que são úteis para a predição. Com a experiência obtida até

agora através da análise de dados deste experimento, podemos afirmar que dados de atraso próximos devem ser redundantes, já que o período de oscilação está em torno de 11 anos e atrasos próximos tendem a ter valores muito próximos. Assim, podemos concluir preliminarmente, que este método não é bom o suficiente para selecionar um conjunto com variáveis úteis para representar o problema.

(3) e. *Use o programa [wrapper_lin_regr.m], fornecendo as mesmas informações dos programas anteriores, de modo a realizar a seleção de variáveis empregando wrapper com backward elimination. Comente os resultados obtidos e compare com o item (3c). Não deixe de apresentar as entradas que produzem o menor erro médio RMS para os 10 conjuntos de validação.*

Utilizando o *wrapper* linear com *backward elimination* oferecido pelo professor, foram obitdos os seguintes resultados contidos na **Figura 13**. Os pontos que possuem texto em vermelho representam as variáveis de prova aleatória.

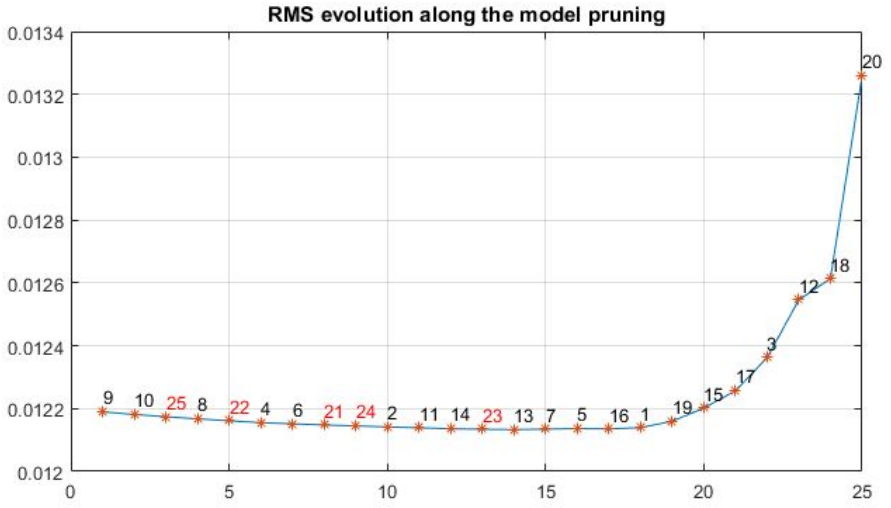


Figura 13: Evolução do erro RMS ao longo da eliminação das variáveis do *wrapper* linear

Ainda, o menor erro registrado, foi obtido quando eliminadas as primeiras 13 variáveis do conjunto, restando o seguinte subconjunto de variáveis:

[13 7 5 16 1 19 15 17 3 12 18 20]

Não é por coincidência, que o menor erro foi obtido, quando a última variável de prova foi eliminada. Na seção de métodos de validação **[GUY03]**, o autor relata que Bi et al. (2003) propôs este método como critério de parada para a eliminação de variáveis. Segundo Bi, variáveis que possuem relevância menor ou igual àquelas obtidas por uma das variáveis falsas, deve ser discartada. Isto se deve ao fato de que a variavel aleatória falsa não melhora mas também piora a qualidade de predição da rede neural, enquanto variaveis redundantes podem piorar a predição, devido ao aumento de complexidade do problema. Portanto, a partir do momento em que a variável de prova é escolhida para ser removida, podemos considerar este um ponto ótimo do calculo de exclusão por diferenças finitas, que já a derivada *J(s+1)* é muito próxima de zero e mais eliminações vão tornar o conjunto de variáveis menos representativo.

(3) f. Apresente o diagrama de barras com a frequência de escolha dos 7 valores definidos para a regularização do modelo linear. É necessário regularizar um modelo linear? Justifique.

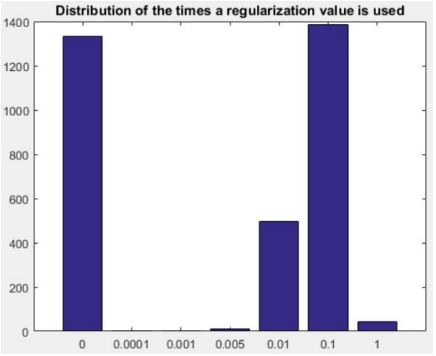


Figura 14: Histograma da utilização do coeficiente de regularização para o *wrapper* linear

Como pode ser observado na **Figura 14**, dentre os valores selecionados para os coeficientes de regularização, quase na maioria das vezes o valor escolhido foi zero. Isto mostra primeiramente, que a matriz H tem quase sempre posto completo, não sendo estritamente necessário o uso dos coeficientes de regularização. No mais, caso estivesse havendo um *overfitting* na predição, os valores mais altos de C seriam na maioria das vezes selecionados, já que penalisariam módulos de pesos sinápticos maiores. Portanto, para este exemplo, não se faz necessário o uso estrito do coeficiente de regularização. Outro fato, é de que para a rede neural linear, faz-se muito dificil ocorrer o *overfitting*, até pela simplificade do mapeamento realizado pela rede neural linear.

(3) g. Explique por que nem sempre as entradas de menor correlação de Pearson são as primeiras a serem podadas na abordagem backward elimination, sendo algumas dessas entradas de baixa correlação até selecionadas para compor a entrada do modelo, ao final.

Como já esclarecido nos ítens anteriores, a correlação de Pearson estabelece a ordem de relevância de modo independente e isolado, de tal modo que muitas vezes seleciona um subconjunto de variáveis relevantes, mas que no entanto podem ser redundantes e assim não possuem poder preditivo suficiente para representar o modelo final. Já os *wrappers* selecionam subconjuntos úteis, podendo excluir muitas variáveis relevantes, mas que são redundantes e não necessárias para a predição do modelo. Assim, filtrar pode ser muitas vezes subótimo para construir um modelo, já que as variáveis podem ser redundantes. Métodos que pontuam variáveis individualmente e independentemente com relação as outras variáveis pecam em determinar a combinação de variáveis que trará a melhor performance.

(3) h. Use o programa [wrapper_nlin_regr.m], fornecendo as mesmas informações dos programas anteriores, de modo a realizar a seleção de variáveis empregando wrapper com backward elimination. O modelo não-linear deve ser uma ELM com um número de neurônios em torno de 100. Comente os resultados obtidos e compare com o item (3e). Não deixe de apresentar as entradas que produzem o menor erro médio RMS para os 10 conjuntos de validação.

Utilizando o *wrapper* não linear ELM com *backward elimination* oferecido pelo professor, foram obitdos os seguintes resultados contidos na **Figura 15**. Os pontos que possuem texto em vermelho representam as variáveis de prova aleatória.

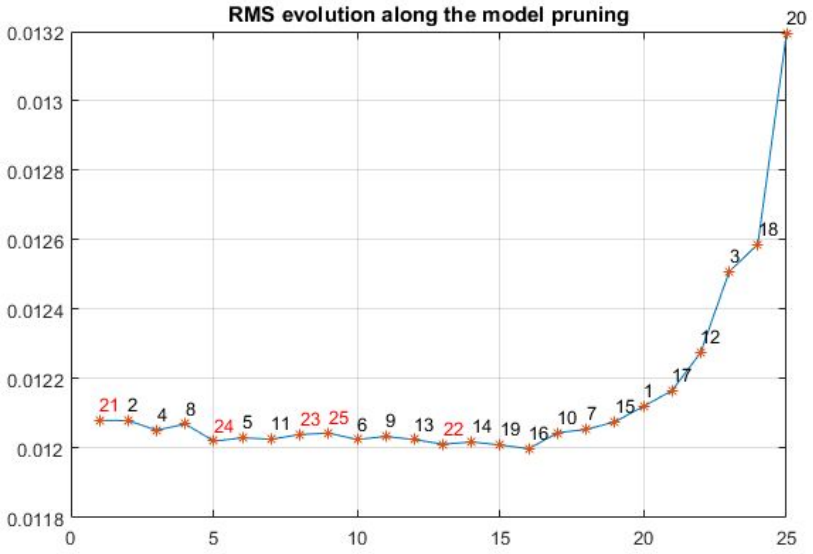


Figura 15: Evolução do erro RMS ao longo da eliminação das variáveis do *wrapper* ELM

Novamente, o subconjunto que possui o menor erro é aquele que já não possui mais as variáveis de prova aleatória introduzidas. Pouco depois da retirada da ultima prova, o erro volta a subir devido a falta de representatividade da variáveis do conjunto para representar o modelo. Para tanto, parar após a retirada da última prova, se mostra uma estratégia bastante eficaz.

Ainda, o menor erro registrado, foi obtido quando eliminadas as primeiras 15 variáveis do conjunto, restando o seguinte subconjunto de variáveis:

[16 10 7 15 1 17 12 3 18 20]

Resumindo as informações obtidas para os dois *wrapper*, obtivemos os seguintes subconjuntos ótimos para os *wrapper* aqui estudados neste relatório, conforme pode ser visto na **Tabela 3**.

Tabela 3: Comparação entre o subconjunto ótimo para o *wrapper* linear e não linear

Wrapper linear	1	3	5	7	12	13	15	16	17	18	19	20
Wrapper não linear		1	3	7	10	12	15	16	17	18	20	

Em comparação com os resultados obtidos pelo *wrapper* linear, pode-se primeramente constatar que o erro final obtido do melhor conjunto é menor para o *wrapper* não linear, embora o subconjunto seja menor em número de dimensões. Isto se deve ao fato de que a rede neural ELM consegue achar correlações não lineares mais complexas entre as entradas e assim elimina as variáveis de forma mais criteriosa e precisa.

Outro fato curioso, é que os dois conjuntos finais são quase identicos, se não pela mudança de poucas variáveis. Dada a pequena de diferença de erro final obtida e a dimensão similar obtida para o conjunto final, pode-se dizer razoável usar um *wrapper* com um modelo de predição linear e depois realmente treinar o modelo através de um modelo não linear mais complexo com as variáveis resultantes. Deste modo, estaria sendo economizado uma quantidade grande de computação,

devido ao fato da rede linear ser muito menos custosa do que a obteção dos pesos para a rede ELM.

Por fim, obtivemos os seguintes dados referentes ao treinamento do *wrapper* ELM:

RMS error considering a model with the 5 inputs with highest correlation
0.0124

RMS error considering a model with the 5 inputs with highest correlation + 5 random inputs
0.0124

Como constatado anteriormente, este resultado mostra que as variáveis aleatórias não aumentam e também não diminuem o erro final obtido da predição do modelo usando o subconjunto selecionado. Ou seja, a partir do momento que eliminamos todas as provas no *wrapper*, significa que já não há mais variáveis a se eliminar de modo que o erro continue a diminuir e podemos então para a eliminação.

(3) i. *Apresente o diagrama de barras com a frequência de escolha dos 7 valores definidos para a regularização do modelo não-linear. Comente*

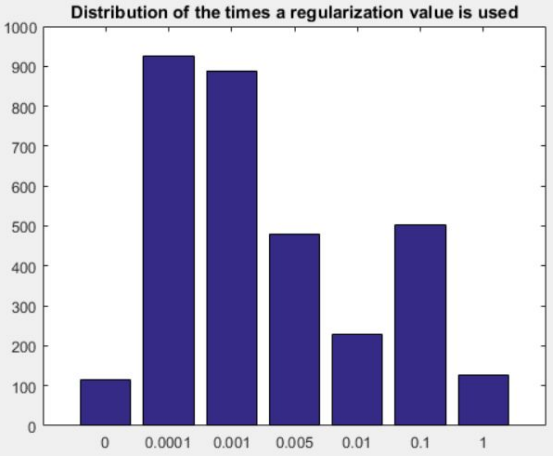


Figura 16: Histograma da utilização do coeficiente de regularização para o *wrapper* ELM

Neste caso, já houve muito mais ocorrências da seleção de coeficientes de regularização mais altos. Ao que tudo indica, estes valores altos estariam prevendo um *overfitting* devido ao alto poder de generalização da rede neural ELM. Para tanto, é sensato contar com o coeficiente de regularização ao treinar o *wrapper* não linear baseado na rede neural ELM.

3. Questão 3

Um classificador usando PCA conseguiria o mesmo desempenho que um classificador usando NLDA para Wine data set?

Embora um classificador linear (PCA) possa obter resultados razoáveis quando há uma boa correlação entre as variáveis e esta correlação é linear, ele ainda possui limitações que se tornam evidentes quando analisamos problemas de maior complexidade. No caso analisado nesta questão, podemos ver na **Figura 17** que a separação das três classes pelo classificador que usa PCA é possível, mas não traz resultados ideais, já que a separação dos pontos de classes diferentes na dimensão reduzida não é tão clara e por isso o classificador terá dificuldade em classificar corretamente as diferentes classes.

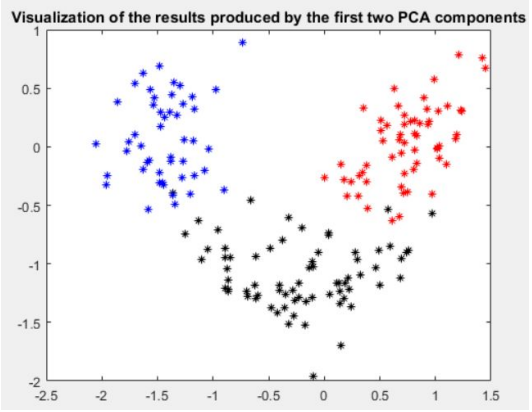


Figura 17: Visualização das diferentes classes na dimensão reduzida produzida pelo encoder PCA.

Assim, podemos concluir que o PCA conseguiria um bom desempenho caso houvesse uma boa correlação entre as variáveis e esta correlação fosse linear. Deste modo temos que o PCA não é muito eficiente para problemas onde a correlação não é perfeitamente linear. Para ilustrar melhor o problema retiramos de **[WAG17]**, como pode ser visto na **Figura 18**, dois casos hipotéticos de diferentes correlações entre duas variáveis. No caso **(a)**, podemos notar que a correlação entre as variáveis é bastante linear e projetar todos os dados sobre o principal hiperplano de projeção não seria uma má idéia, já que representaria muito bem a dimensão original. No entanto, vemos que para o caso **(b)**, embora ainda haja uma clara correlação entre as duas variáveis, o PCA não seria capaz de predizer essa correlação através de um hiperplano. Desto modo, através de uma análise de componentes principais usando uma rede MLP seria possível obter com até bastante facilidade uma correlação não linear entre as variáveis e assim projetar os dados nesta hipersuperficie para conseguir uma dimensão reduzida mais relevante e útil. Vale lembrar, que como toda rede MLP, estamos sujeitos também a *under* e *overfitting* ao estabelecer a hipersuperfície sobre os dados de entrada. Assim, deve-se controlar o ajuste adequado com métodos de validação.

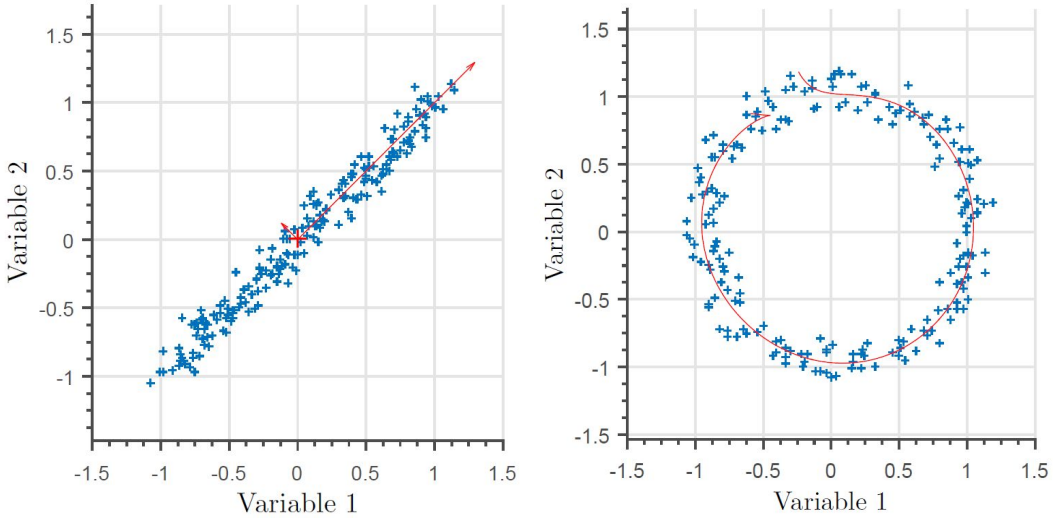


Figura 18: **(a)** PCA na correlação linear entre variaveis 1 e 2 **(b)** MLP usada para correlação não linear entre as variaveis 1 e 2

Deste modo, prosseguimos para o próximo passo, realizando a classificação para o mesmo problema mas agora usando o NLDA. Como esperado, na **Figura 19** vemos que o encoder não linear MLP treinado por 100 épocas se mostrou muito mais eficiente em reduzir a dimensão do problema de tal forma que a classificação entre as diferentes classes se tornou posteriormente mais facil. Podemos ver também que para o mesmo número de épocas, diferentes execuções resultaram em uma certa diferença nos dados gerados pelo encoder. Isso ocorre pois o valor final dos pesos sinápticos da rede neural é influenciada pelo chute inicial e ocorre dos pesos estagnarem em algum mínimo local ou ainda o número de épocas não foi suficiente para se chegar em um bom mínimo durante o processo de otimização. No entanto, embora os pesos sinápticos finais da rede neural sejam diferentes, a execução com o mesmo número de épocas resultaram em um erro quadrático médio muito parecido.

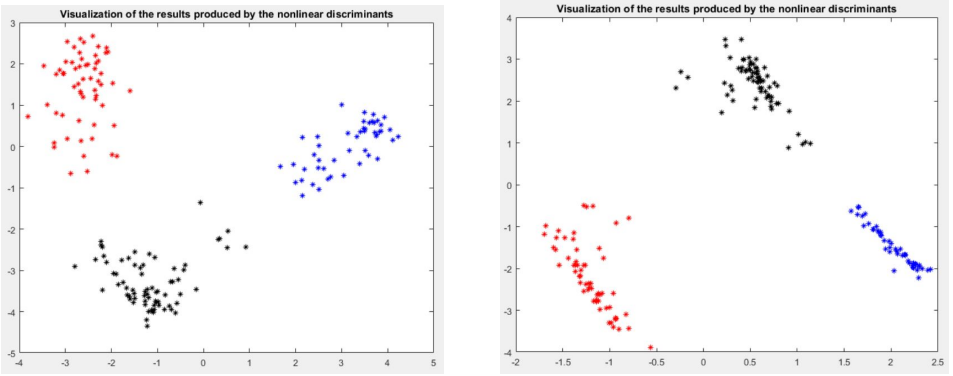
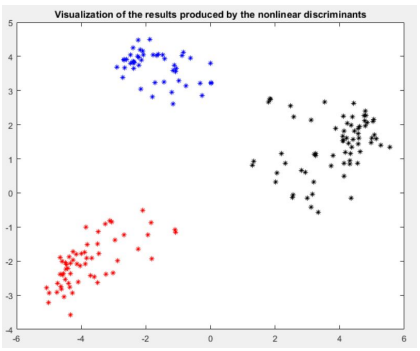


Figura 19: Visualização das diferentes classes na dimensão reduzida produzida pelo encoder NLDA. Ambas imagens foram produzidas com mesmo número de iterações. *RMSE = 0.03351, No. of iterations = 100*

Podemos ver também nas **Figuras 20**, que ao executarmos o processo de ótimização dos pesos sinápticos por mais épocas, foi possível obter erros quadráticos médios *RMSE*, ainda menores.



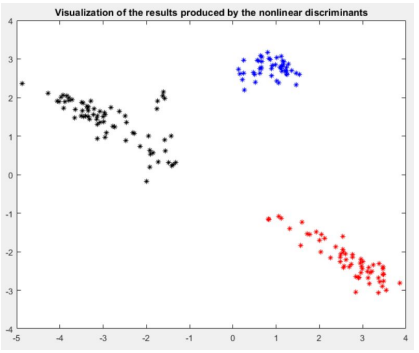


Figura 20: **(a)** *RMSE* = 0.00001, *No. of iterations = 1000* **(b)** *RMSE = 0.00012* *No. of iterations = 500*

Em resumo, uma breve análise do erro quadrático médio relativo ao número de épocas, pode ser encontrado na **Tabela 4** abaixo.

Tabela 4: Análise do erro de validação em relação ao número de iterações o problema do Wine dataset

Número de épocas	Erro médio quadrático
100	0.03351
300	0.00137
500	0.00012
1000	0.00001

Assim, concluimos que para o problema do *Wine data set*, o uso de uma correlação não linear entre as variáveis de entrada para se reduzir a dimensão do problema é muito mais adequada do que a correlação linear feita pelo PCA.

Vale ainda ressaltar, que melhores resultados teriam sido obtidos com o uso de *Autoencoders*. Conforme sugerido em **[MAT17]**, diversas camadas de *Autoencoders* ligadas ao fim à uma camada de classificador e treinadas todas separadamente, generalizam melhor o problema e trazem melhores resultados. Uma breve ilustração da configuração da rede neural, que no caso já podemos chamar de *Deep Learning*, para o problema do MNIST pode ser vista na **Figura 21**. Deste modo, também poderiamos ter usado algo muito similar na resolução do problema para o *Wine data set*.

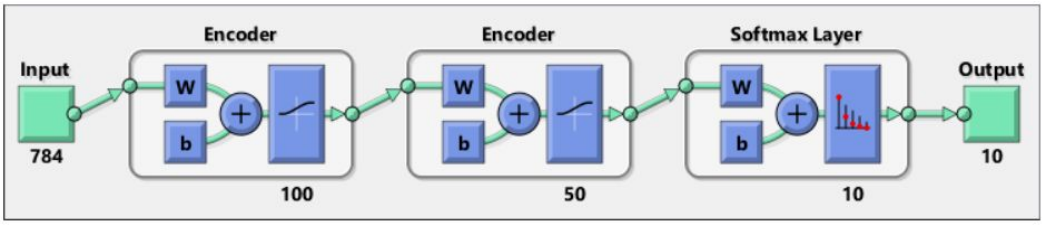


Figura 21: Estrutura da rede neural MLP composta por 2 camadas de *autoencoders* e uma final de classificação para o problema MNIST **[MAT17]**

4. Referencias

[TAP14] Tapson, J. (2014). *Explicit Computation of Input Weights in Extreme Learning Machines*. Proceedings of ELM-2014 Volume 1 pp 183-191.

[ZXU14] Z. Xu, M. Yao. (2014). *A Fast Incremental Method Based on Regularized Extreme Learning Machine*. Proceedings of ELM-2014 Volume 1 pp 15-30.

[GUY03] Guyon, I. (2003). *An Introduction to Variable and Feature Selection*. Journal of Machine Learning Research 3 (2003) 1157-1182.

[WAG17] Wagner, S. (2017). *Parametric Analysis of a Lateral Control System for Simulations of Highly Automated Driving*. Technical University of Munich.

[WAN13] Wan, L. (2013). Regularization of Neural Networks using DropConnect.

[MAT17] Mathworks Documentation. *Train Stacked Autoencoders for Image Classification*. Acessado em 01/10/2017 às 14:10.