Audi
Issa Albtoush

PhD Thesis

New neural networks based on Extreme Learning Machine

Santiago de Compostela, 2022

Doctoral Programme in Information Technology Research

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

TESE DE DOUTORAMENTO

NEW NEURAL NETWORKS BASED ON EXTREME LEARNING MACHINE

Audi Issa Albtoush

ESCOLA DE DOUTORAMENTO INTERNACIONAL DA UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

PROGRAMA DE DOUTORAMENTO EN INVESTIGACIÓN EN TECNOLOXÍAS DA INFORMACIÓN

SANTIAGO DE COMPOSTELA
2022

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

DECLARACIÓN DO AUTOR DA TESE

Don Audi Issa Albtoush

*Presento a miña tese, titulada **New neural networks based on Extreme Learning Machine**, seguindo o procedemento adecuado ao Regulamento, e declaro que:*

1. *A tese abarca os resultados da elaboración do meu traballo.*
2. *De ser o caso, na tese faise referencia ás colaboracións que tivo este traballo.*
3. *Confirmo que a tese non incorre en ningún tipo de plaxio doutros autores nin de traballos presentados por min para a obtención doutros títulos.*
4. *A tese é a versión definitiva presentada para a súa defensa e coincide a versión impresa coa presentada en formato electrónico.*

E comprométome a presentar o Compromiso Documental de Supervisión no caso de que o orixinal non estea na Escola.

En Santiago de Compostela, 30 de xaneiro de 2022

Asdo. Audi Issa Albtoush

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

AUTORIZACIÓN DO DIRECTOR DA TESE

Don Manuel Fernández Delgado, Profesor Titular da Área de Ciencias da Computación e Intelixencia Artificial, Departamento de Electrónica e Computación, Universidade de Santiago de Compostela en condición de **director** da presente tese, titulada **New neural networks based on Extreme Learning Machine**

INFORMA:

Que a presente tese correspóndese co traballo realizado por ***Don Audi Issa Albtoush****, baixo a miña dirección, e autorizo a súa presentación, considerando que reúne os requisitos esixidos no Regulamento de Estudos de Doutoramento da USC, e que como director desta non incorre nas causas de abstención establecidas na Lei 40/2015.*
De acordo co indicado no Regulamento de Estudos de Doutoramento, declara tamén que a presente tese de doutoramento é idónea para ser defendida en base á modalidade Monográfica con reprodución parcial de publicacións, nos que a participación do doutorando foi decisiva para a súa elaboración e as publicacións se axustan ao Plano de Investigación.

En Santiago de Compostela, 30 de xaneiro de 2022

Asdo. Manuel Fernández Delgado
Director da tese

Dedication

I am dedicating this thesis to two beloved people who have meant and continue to mean so much to me. Although they are no longer of this world, their memories continue to regulate my life. To my paternal uncle Dr. Abdallah Albtoush and my maternal uncle Amer Althiabat, I will never forget you.

This work is also dedicated to my parents, who have always loved me unconditionally and whose good examples have taught me to work hard for the things that I aspire to achieve.

To my beloved brothers and sisters;
To all my family, the symbol of love and giving,
To my friends who encourage and support me,
To All the people in my life who touch my heart, I dedicate this research

Acknowledgments

This work has received financial support from the Consellería de Cultura, Educación e Universidade (accreditation 2019-2022 ED431G-2019/04) and the European Regional Development Fund (ERDF), which acknowledges the CiTIUS - Centro Singular de Investigación en Tecnoloxías Intelixentes da Universidade de Santiago de Compostela as a Research Center of the Galician University System.

30 de xaneiro de 2022

Contents

1 Introduction **9**

1.1 Motivation and scope . . . 9

1.2 Structure of the thesis . . . 11

1.3 Theoretical foundations of ELM . . . 12

1.4 Variants of the extreme learning machine . . . 16

1.5 Large scale datasets and big data . . . 17

1.6 Applications of ELM networks . . . 19

2 Quick extreme learning machine for large-scale classification **21**

2.1 Extreme learning machine . . . 23

2.2 Number of hidden neurons . . . 25

2.3 Selection of H in QELM . . . 26

2.4 Calculation of the output weights in QELM . . . 29

2.5 The QELM algorithm . . . 32

2.6 Results and discussion . . . 35

3 Extreme learning machine with confidence interval based bias initialization **45**

3.1 Confidence interval and confidence level . . . 47

3.2 Confidence random bias extreme learning machine . . . 50

3.3 Confidence random bias-singular value decomposition ELM . . . 50

3.4 Classification results . . . 52

3.5 Regression results . . . 57

4 Quick hidden layer size tuning in extreme learning machine for classification **63**

4.1 Moving average extreme learning machine . . . 64

4.2 Exponential moving average extreme learning machine 67
4.3 Divide-and-conquer extreme learning machine 67
4.4 Experimental methodology 69
4.5 Comparison of MA-ELM with ELM and CELM 72
4.6 Comparison of EMA-ELM with ELM and CELM 73
4.7 Comparison of DC-ELM with ELM and CELM 81
4.8 Discussion 84

5 Conclusions **85**

A Listado de publicacións **89**

Bibliography **91**

Resumo en galego

A presente tese de doutoramento pertence ao ámbito da intelixencia artificial, e máis concretamente ao campo das redes neuronais. Un modelo de rede neuronal especialmente estudado nos últimos anos é a *extreme learning machine* (ELM). Esta rede despertou particular interese no ámbito da aprendizaxe automática pola súa simplicidade, xa que a diferencia das redes neuronais clásicas, e das modernas redes profundas (*deep learning*), a rede ELM é sumamente sinxela tanto na súa estructura como no seu algoritmo de aprendizaxe. A súa estructura está formada por unha capa de neuronas de entrada, unha soa capa de neuronas ocultas, a diferencia doutras redes multicapa, e unha capa de neuronas de saída. Esta simplicidade está asociada tamén a unha extrema velocidade no entrenamento, a diferencia dos outros modelos neuronais, que usualmente son moito máis lentos. Específicamente, os pesos a entrenar son somentes os pesos de conexión entre as capas de entrada e oculta, denominados pesos de entrada, e máis os pesos de conexión entre as capas oculta e de saída, denominados pesos de saída. Pola outra banda, as neuronas da capa oculta posúen tamén unhas polarizacións (*offsets* en inglés) con valores entrenábeis, e usan unha función de activación, que dota á rede neuronal da capacidade para aprender funcións non lineares. Esta función pode adopar distintas formas, entre elas a sigmoide, que é a máis clásica, pero tamén a tanxente hiperbólica, senoidal, función umbral, base triangular, base radial e cosenoidal, entre outras.

Tanto a simplicidade como a velocidade das redes ELM débense a que os pesos de entrada se inicializan aleatoriamente, mentres que os pesos de saída se calculan usando un simple producto matricial e calculando a pseudo-inversa, ou inversa xeralizada, dunha matriz, aínda que este último procedemento non é tan eficiente computacionalmente, sobre todo para matrices grandes. Deste modo, o entrenamento da ELM ten unha expresión analítica pechada, é decir, non require procesos iterativos de cálculo numérico que poden ser lentos ou conducir a configuracións de aprendizaxe non óptimas. Pola outra banda, demostrouse matemáticamente que, similarmente ás redes neuronais máis empregadas, a ELM é un aproximador universal, é decir, que ten capacidade para aprender calquera función cun certo grado de precisión. Dado que esta rede é un algorimo de aprendizaxe xeral, tense aplicado a todo tipo de problemas de aprendizaxe automática. A diferencia de outras redes neuronais, nas que a clasificación constitúe a súa principal aplicación, seguindo á literatura a rede ELM aplícase principalmente a problemas de regresión (máis do 40% das aplicacións), seguida pola clasificación (31%) e outros como aprendizaxe de características (13%), compresión de datos (10%) e agrupamento ou *clustering* (8%). A literatura desenvolveu tamén distintas variantes desta rede, incluíndo versións incrementais e regularizadas, secuenciais *on-line*, comités, ELM ponderada e ELM restrinxida. Ademáis, esta rede combinouse coa lóxica *fuzzy*, con algoritmos xenéticos, *particle swarm optimization*, algoritmos evolucionarios e redes profundas, entre elas o ELM *autoencoder*. A literatura inclúe estudos comparativos da ELM coa support vector machine (SVM), e incluso se desenvolveu unha versión *Twin ELM* inspirada na popular *Twin SVM*. Importantes esforzos se fixeron tamén para aplicar a ELM a datos grandes, aínda que limitadas a problemas de clasificación con poucos centenares de miles de datos, e para extendela a plataformas *Big Data* como Spark e MapReduce.

A pesar da súa simplicidad e potencia de aprendizaxe, a rede ELM ten dous inconvintes principais, especialmente con datos grandes, como acontece por exemplo cando hai moitos datos ou estos son dunha dimensionalidade moi elevada. O primeiro inconvinte está motivado, similarmente ás outras redes neuronais, pola necesidade de sintonizar o número de neuronas da capa oculta, en adiante referenciado como H, valor que resulta de grande importancia porque inflúe enormemente na capacidade de aprendizaxe da rede. Por este motivo, dependendo da complexidade e do tamano dos datos pode ser necesario un número elevado de neuronas ocultas. Nembargantes, un número excesivo, aínda que permite aprender cunha precisión moi elevada o conxunto de entrenamento, con frecuencia conduce a problemas de sobreaprendizaxe de modo que a ELM funciona mal sobre datos de teste, é decir, datos non usados durante o entrenamento. Resulta, pois, necesario atopar un valor axeitado para H, e isto consíguese usualmente probando con distintos valores e seleccionando o que proporciona á rede ELM unha meirande calidade, avaliada usando as medidas correspondente para clasificación, regresión, etc. O problema é que este procedemento esixe entrenar a rede ELM moitas veces, ralentizando moito a súa operación. O segundo inconvinte da rede ELM débese á necesidade de efectuar a pseudo-inversión dunha matriz, xa que esta matriz é con frecuencia de grande tamano. Concretamente, canto máis grandes son os datos, máis grande é esta matriz. Neste caso en que a matriz é grande, a pseudo-inversión é moi lenta, e incluso pode ser imposible, é decir, non se pode executar porque conduce a un erro de execución, se a matriz non cabe na memoria RAM dispoñíble. Este inconvinte provoca que a ELM sexa difícilmente aplicable a un problema con moitos datos, e polo tanto reduce moito a escalabilidade destas redes.

Ambos problemas foron diagnosticados na literatura, e propuxéronse diversas solucións, que analizamos nesta tese. Desafortunadamente, estas solucións normalmente non son satisfactorias porque incrementan a complexidade destas redes, desperdiciando a súa principal vantaxe, que é a súa simplicidade e velocidade, e dificultando aínda máis a súa aplicación a problemas grandes. Outras solucións están orientadas non a aumentar a súa velocidade, senón a acadar mellores resultados, usualmente a costa novamente dun aumento na súa complexidade coa consecuente ralentización da súa execución (entrenamento e/ou test). A tese actual presenta novos algoritmos baseados nestas redes ELM, deseñados coa premisa de manter a súa simplicidade, grazas á inicialización aleatoria dos pesos de entrada e ao uso da matriz pseudo-inversa para calcular os pesos de saída, e a súa velocidade, e co obxectivo de aportar solucións para as mencionadas deficiencias pensando especialmente nos datos grandes. O capítulo 1 da actual memoria describe estos obxectivos e introduce a teoría da ELM, xunto co estudo das súas principais versións e aplicacións, cun especial énfase nas asociadas a datos grandes.

A primeira versión proposta, no capítulo 2 desta tese, é a quick extreme learning machine (QELM), unha versión eficiente da ELM para problemas de clasificación de grande tamano [12]. Esta rede aporta solucións aos dous inconvintes mencionados previamente. Para seleccionar un valor axeitado do número H de neuronas na capa oculta, evítase a necesidade de repetir o entrenamento da rede para distintos valores, o cal resulta inaceptable con problemas grandes pola súa lentitude. Con este fin, estudouse como se comporta o valor óptimo de H en función do número N de datos, de modo que se poda predecir H directamente a partir de N. Para descartar a influencia da compoñente aleatoria na calidade da clasificación, empregouse para tódolos valores de H a mesma inicialización aleatoria dos pesos de entrada da rede ELM. O estudo considerou unha colección de conxuntos de datos de clasificación, availando a calidade da rede ELM sobre os conxuntos de entrenamento e teste. O resultado do experimento indica que a calidade da rede aumenta con H/N para valores baixos de H/N, pero se reduce cando $H/N \to 1$ como consecuencia da sobre-aprendizaxe, xa que a calidade sobre o entrenamento aumenta con H/N, pero diminúe a calidade sobre o conxunto de teste. A partir destos resultados pódese concluir que o valor óptimo de H/N é relativamente constante e non demasiado elevado, o cal é bo para datos con N grande, nos que o H óptimo non resulta ser tan elevado. Polo tanto, estos resultados permiten definir unha expresión de H crecente en función de N. Ademáis, isto permite seleccionar un valor de H sen repetir o entrenamento da rede, nin tampouco efectuar cálculos complexos sobre os datos, xa que calquera das dúas opcións pode ser extremadamente lenta con datos grandes. Dado que o valor óptimo de H é crecente co número N de datos, débese ter coidado xa que o H óptimo seleccionado pode ser excesivamente elevado se N é alto, e isto podería imposibilitar a pseudo-inversión por tratarse de matrices extremadamente grandes. Para evitar este tipo de problemas, o algoritmo proposto QELM establece que H estea acotado para N grande, de modo que o tamano da matriz sobre a cal efectuar a pseudo-inversión teña un tamano sempre aceptable para efectuar a pseudo-inversión por moi elevado que sexa N.

O número N de datos de entrenamento tamén inflúe no tamano da matriz da cal se debe calcular a pseudo-inversa. Isto é un inconvinte, xa que cando N medra (por exemplo, con datos grandes) chegará un punto en que a pseudo-inversa xa non se pode calcular por limitacións de potencia de cálculo ou de almacenamento da matriz. Para evitar este problema, QELM substitúe os datos orixinais por unha colección de prototipos ou promedios calculados a partir de aqueles. O punto importante aquí é que o número de prototipos está acotado, de modo que o tamano da matriz se manteña en límites manexables para o seu almacenamento e a pseudo-inversión matricial. Deste modo, a colección de prototipos elabórase como un "resumo" do conxunto completo de datos, empregando un algoritmo de agrupamento que que os prototipos representan axeitadamente aos datos. Trátase de substituír na matriz a invertir os datos orixinais polo seu resumo, co obxectivo de que estos últimos permitan á rede ELM aprender a información fundamental dos datos pero sen procesalos todos, o cal non sería factíble computacionalmente. A elaboración dos prototipos efectúase de forma eficiente para evitar que o seu custe computacional compense as vantaxes que o uso de prototipos no canto dos datos ten sobre a velocidade de entrenamento.

O traballo experimental avalía a calidade de clasificación e o tempo de execución de QELM comparándoos coas mellores técnicas da literatura, incluíndo a rede ELM clásica, a support vector machine (SVM) con núcleo gausiano, e a SVM linear, sobre conxuntos de datos pequeños (menos de 15000 datos) e grandes (máis de 15000 datos ata 31 millóns de datos con dimensionalidades ata 30000), máis que en calquera outro traballo atopado na literatura. Os resultados da rede QELM mostran calidades na clasificación moi similares aos da ELM, que non están moi alonxados dos acadados pola SVM, con tempos de entrenamento ata 36 e 154 veces inferiores aos da ELM e a SVM nos datos pequenos. Nos conxuntos de datos grandes, a ELM non se pode executar no 41% dos datos máis grandes, mentres que QELM pódese executar en todos eles sendo 14 veces máis rápida que a SVM linear, a única que se pode executar nestes datos porque a SVM gausiana tarda un tempo inaceptable en entrenar.

Ademáis, QELM acada unha calidade de clasificación moi similar á ELM nos datos nos que esta se pode executar, e moi superior á calidade da SVM linear. Pola outra banda, a QELM pode ser executada en datos arbitrariamente grandes sen producirse erros de execución, aínda que o tempo requerido obviamente se incrementa co tamano dos datos porque a creación dos prototipos require máis tempo cantos máis datos hai. Nembargantes, os tempos requeridos polo entrenamento (descartando os prototipos) e o test da QELM non medran significativamente a partir de certo tamano de datos, xa que o tamano da matriz a invertir non medra máis aló dos seus límites pre-definidos. A rede QELM non ten hiper-parámetros que requiran sintonización, xa que os seus valores non afectan significativamente á calidade dos resultados. Finalmente, hai que destacar que esta proposta mantén a simplicidade da rede ELM clásica na súa estructura e entrenamento, e tamén a súa velocidade tanto para datos pequenos como grandes.

O segundo algoritmo, proposto no capítulo 3 desta tese e publicado no artigo de congreso [13], afecta á xeración dos valores aleatorios para as polarizacións das neuronas ocultas. Tanto os pesos de entrada como as polarizacións das neuronas ocultas son inicializados na rede ELM de modo aleatorio. Esta aleatoriedade pode influír nos resultados, dependendo do rango no que se atopen os valores aleatorios usados. De feito, un rango incorrecto podería provocar resultados pobres. O método proposto, denominado confidence-random-based extreme learning machine (CDB-ELM), explora o uso de valores aleatorias para estas polarizacións que teñan en conta os valores específicos dos datos de entrada (D), ou ben dos pesos de entrada (W), ou ben do producto (WD) dos datos polos pesos, dando lugar a tres versións distintas de CRB-ELM. O método para a xeración das polarizacións fai uso dos conceptos estatísticos de nivel de confianza e intervalo de confianza. O nivel de confianza, denotado por F, é a probabilidade, en %, de que os valores aleatorios da polarización se atopen fóra do intervalo de confianza. Este intervalo está definido polos seus límites inferior e superior, que se calculan mediante unha serie de fórmulas estatísticas ben coñecidas na literatura. Estas fórmulas teñen en conta a media e a desviación típica dos datos, pesos ou datos multiplicados polos pesos, e empregan tamén a fórmula acumulativa da distribución normal definida pola devandita media e desviación, xunto coa probabilidade definida por F. Logo de xerar estas polarizacións aleatorias, execútase o entrenamento da ELM clásica.

A experimentación compara a rede proposta CRB-ELM coa ELM sobre conxuntos de datos de clasificación e regresión. O método CRB para seleccionar o rango das polarizacións aleatorias aplícase tamén á descomposición en valores singulares (*singular value decompo-*

sition ou SVD, polas súas siglas en inglés) da matriz de datos, dando lugar á version CRB-SVD-ELM, que se compara coa *base projection machine* (BPM), un conocido algoritmo de aprendizaxe relacionado coa ELM que tamén utiza SVD. A rede CRB-SVD-ELM substitúe os pesos de entrada aleatorios polos valores dunha das tres matrices resultantes de aplicar a técnica SVD aos datos de entrada, empregando un número *H* de neuronas ocultas igual ao rango de outra das matrices xeradas pola SVD. Os resultados revelan que a CRB-ELM e a CRB-SVD-ELM melloran sensíblemente os de ELM, tanto en termos de acerto para clasificación como de erro cadrático medio en conxuntos de datos de regresión. Comparando as tres versions de CRB-ELM, que inicializan as polarizacións usando o intervalo de confianza para datos (D), pesos (W) e datos por pesos (WD), a versión que emprega os datos exclusivamente acada un acerto claramente superior ás outras dúas, obtendo o acerto máximo en tódolos datos aínda que nalgúns deles algunha das outras dúas versións tamén acadan o mesmo acerto. A versión de datos tamén é superior á ELM clásica na grande maioría dos conxuntos de datos. Pola outra banda, comparando a rede proposta CRB-SVD-ELM coa rede BPM a primeira supera á segunda en termos de acerto na grande maioría dos conxuntos de datos de clasificación, obtendo un acerto promedio sobre tódolos datos lixeiramente superior á BPM. Nos datos de regresión, a rede CRB-SVD-ELM acada o menor error cadrático medio en tódolos conxuntos de datos, aínda que a BPM tamén acada o mesmo erro en aproximadamente a metade deles, cun erro promedio lixeiramente superior a CRB-SVD-ELM. Novamente, hai que destacar que a modificación algorítmica introducida por CRB-ELM, consistente no uso do intervalo de confianza para a inicialización aleatoria das polarizacións nas neuronas ocultas, non incrementa a complexidade da ELM e respeita as súas propiedades de simplicidade e velocidade, mellorando aínda así a súa calidade na aprendizaxe mediante unha elección axeitada do intervalo de xeración de valores aleatorios, neste caso para as polarizacións das neuronas da capa oculta.

O capítulo 4 aborda unha terceira modificación algorítmica da ELM en problemas de clasificación, neste caso para acelerar o proceso de selección dun número *H* axeitado de neuronas ocultas reducindo o número de valores de *H* empregados, ou equivalentemente o número de veces que se executa o entrenamento da ELM, e polo tanto o tempo de entrenamento-sintonización. Para isto, empréganse os conceptos estatísticos de "promedio móvil' (*moving average* ou MA, polas súas siglas en inglés), "promedio móvil exponencial" (EMA) e "divide e vencerás" (*divide-and-conquer* ou DC, polas súas siglas en inglés), unha estratexia sumamente empregada en moitos campos da intelixencia artificial e da informática en xeral. Estas tres estratexias dan lugar ás versións MA-ELM, EMA-ELM and DC-ELM. A técnica de MA consiste en estimar o acerto do clasificador ELM sobre o conxunto de entrenamento para un valor novo de H a partir dunha serie de n valores previos do acerto para valores anteriores de H, sendo n o chamado "ancho da ventá" de cálculo do promedio móvil. Os valores de H deben estar equiespaciados entre si. O acerto estimado compárase co acerto verdadeiro. Se ambos valores son suficientemente cercanos, dentro dunha tolerancia pre-definida, a partir deste momento os valores do acerto calcularanse usando o método MA, evitándose novos entrenamentos da ELM e aforrándose o tempo correspondente. Se, polo contrario, a diferencia entre os valores estimado e verdadeiro supera a tolerancia, o acerto para o seguinte valor de H será novamente estimado polo MA e calculado entrenando a ELM, sen aforro no tempo. Isto repítese ata que ambos valores son cercanos, momento a partir do cal sempre se usa o MA, ou ata que se rematan os valores de H, caso no que o método MA non reduciría nada o tempo de sintonización comparado coa ELM. O obxectivo é, obviamente, evitar o meirande número posible de entrenamentos da ELM para reducir o tempo requerido, e isto require que o MA prediga axeitadamente o acerto o antes posible. O algoritmo EMA-ELM é similar ao MA-ELM, pero usando o EMA, que concede ao acerto máis recente un peso meirande que os acertos anteriores no cálculo do promedio estatístico. Tanto MA como EMA reducen de modo importante o número de entrenamentos, xa que a partir do momento en que os acertos estimado e calculado están cercanos coa tolerancia usada non se volve entrenar a ELM, porque tódolos acertos se estiman usando MA ou EMA. Non obstante, é posible que esta coincidencia entre acerto estimado e calculado non se produza para ningún valor de H, caso no cal non se aforraría tempo ningún. O terceiro método, DC-ELM, divide recursivamente o conxunto de valores de H en dous subconxuntos, calculando os acertos da ELM usando os valores máximos de H en ámbolos dous subconxuntos e seleccionando o que proporciona un acerto meirande. Este subconxunto divídese novamente e calcúlanse os acertos correspondentes aos seus valores máximos, repetíndose o proceso recursivamente ata que se acada un subconxunto cun so valor de H. Este valor é o seleccionado como óptimo, evitando un número importante de entrenamentos da rede ELM e reducindo substancialmente o tempo de execución en comparación coa sintonización clásica *grid-search*. Os resultados destas tres redes, comparadas coa ELM clásica e coas variantes da *Constrained ELM* sobre conxuntos de datos *benchmark* de clasificación mostran importantes reduccións no tempo de entrenamento-sintonización de hiper-parámetros, situadas nos rangos 80-98% no caso de MA-ELM e EMA-ELM, e 82-84% para DC-ELM, que son entre 5 e 50 veces máis rápidos que a ELM clásica acadando o mesmo

nivel de acerto.

O traballo futuro inclúe extender as capacidades de QELM a conxuntos de datos cun número de entradas aínda meirande; integrar as tres versións de ELM propostas; e profundizar na optimización do cálculo da matriz pseudo-inversa de matrices grandes, o cal constitúe aínda o principal inconvinte das redes ELM para extender o seu uso a problemas xerais de aprendizaxe automática.

Palabras chave: aprendizaxe automática, redes neuronais, *extreme learning machine* (ELM), *large-scale datasets*.

CHAPTER 1

INTRODUCTION

1.1 Motivation and scope

Due to technological development, new data are created every moment and their size and dimensions grow day by day. Thus, it is important to develop efficient and effective "machine learning" methods capable of dealing with this development and diversity in data. These methods are necessary to mitigate the growing tension between machine learning and data to extract useful knowledge and insights from this wealth of information.

In 2004, the extreme learning machine (ELM) appeared as a singular tool in machine learning. The ELM is a type of single-layer feed-forward neural network characterized by: 1) random input weights and bias of the hidden layer neurons; and 2) output weights computed by multiplying the Moore Penrose pseudo-inverse, or generalized inverse, of the hidden layer activation matrix, by the true output matrix. Thus, this network issues a closed-form computing of the output connection weights, that provides high effectiveness, quick training and ease of implementing. That made it an interest and attract algorithm for artificial intelligence developers, compared to existing feed-forward networks where the training required an iterative numeric calculation with a heavy computational cost both in terms of time and memory requirements. Before the ELM, the neural network research community was stucked after almost two decades of slow iterative training methods proposed to refine the backpropagation training algorithm (1986) and to avoid its problems, mainly the fall in local sub-optimal error minima.

However, ELM has left us many inquiries that require to research and improve the performance of the algorithm. From a theoretical point of view, it has proven to be a universal approximator. The randomness of input weights and biases of the neurons in the hidden layer, combined with the pseudo-inverse based closed-form training, should be sufficient to solve any approximate problem and to deal with large scale datasets. However, from a practical point of view proper care must be taken using ELM to get the proper performance and good generalization abilities. The focus of this thesis is on developing effective methods capable of improving ELM performance to deal with the challenges posed by large scale datasets, while keeping the ELM contributions: random initialization of the input weights, pseudo-inverse-based closed-form calculation of the output weights, high simplicity and speed at least for small datasets. In order to achieve this objective, the contributions of the current thesis are along three directions:

1. **Quick extreme learning machine for large scale classification**. The ELM became popular because it uses a fast closed-form expression for training that minimizes the training error with good generalization ability to new data. It requires the tuning of the hidden layer size, that is a hyper-parameter with strong influence on performance, and the calculation of the pseudo-inverse of the hidden layer activation matrix for the whole training set. With large scale classification problems, the computational overload caused by tuning becomes not affordable, and the activation matrix is extremely large, so the pseudo-inversion is very slow and eventually the matrix will not fit in memory. This thesis proposes quick extreme learning machine (QELM), that is able to manage large classification datasets because it: 1) avoids the hyper-parameter tuning by using a bounded estimation of the hidden layer size from the data population; and 2) replaces the training patterns in the activation matrix by a reduced set of prototypes in order to avoid the storage and pseudo-inversion of large matrices. While ELM or even the linear SVM can not be applied to some of the large datasets considered in this thesis (up to 31 million data, 30,000 inputs and 131 classes), QELM can be executed on them spending reasonable times (less than 1 hour) in general purpose computers without special software nor hardware requirements and achieving performances very similar to ELM. This work has been published in the journal paper [12].

2. **Extreme learning machine with confidence interval based bias initialization**. The random initialization of the input weights and biases of the classical ELM network introduces high sensitivity to input perturbations and results in poor network stability. In this thesis, we propose the confidence random bias ELM (CRB-ELM), that inherits

the randomness of the ELM for bias tuning but based on confidence interval and confidence level calculated from the training data. The experimental comparison of CRB-ELM to the classical ELM and the base projection vector machine (BPVM) reports that our proposal achieves higher performance and is more stable both in classification and regression benchmark datasets. This proposal has been published in the conference paper [13].

3. **Quick hidden layer size tuning in ELM for classification problems**. During hyper-parameter tuning, the ELM suffers of time-consuming because several values of the hidden layer size must be tried. The training speed of ELM critically depends on the hidden layer size, and to try many size values, that can be large and lead to slow trainings, is undesirable for real applications and not acceptable for a real-time response. This thesis proposes three methods, named MA-ELM, EMA-ELM, and DC-ELM, which use statistical criteria (moving average, exponential moving average and divide-and-conquer strategy, respectively) to optimize the hidden layer size in ELM so quickly as needed. Compared with the original ELM and different "constrained extreme learning machine" versions (CELMs), the MA-ELM, EMA-ELM and DC-ELM achieved a percentage reducing time up to 98% as well as better generalization ability.

1.2 Structure of the thesis

The current memory describes the topics and theory relevant for the ELM framework, highlighting the results achieved in the publications. The remainder of the current chapter describes the ELM and its theoretical foundations, compiles the main ELM versions and those variants specifically devoted to large scale datasets, alongside with the most relevant applications of these networks. Afterwards, chapter 2 discusses quick extreme learning machine for large scale datasets, chapter 3 describes ELM with confidence interval based bias initialization, and chapter 4 describes MA-ELM, EMA-ELM and DC-ELM, that reduce the computational time in the ELM by quick optimization of the hidden neurons in classification problems. Finally, chapter 5 presents the conclusions of the current thesis and the future work.

1.3 Theoretical foundations of ELM

The single-layer feedforward neural network [44] is one of the most popular class of neural networks. It consists of one hidden layer of neurons that receives stimuli from the external environment and one output layer that sends the network's output to the external environment. These networks are distinguished by their simple structure consisting of one input, one hidden, and one output layer, and have proven to be effective and have a wide range of applications [111]. There are three main families of methods used to train feedforward networks: gradient-descent, e.g. backpropagation [99]; standard optimization [98], e.g. support vector machine (SVM); and least-square based, e.g. radial basis function (RBF) network learning [79]. The ELM was originally developed as a type of single-hidden-layer feedforward neural networks (SLFNs). Opposed to the existing methods to train these networks, that used iterative training algorithms such as back-propagation, the ELM combines randomization of input weights and least-squares optimization to calculate the output weights. This network tends to have the smallest training error and the smallest norm of output weights according to the neural network theory [15]. The nonlinear activation functions in the hidden layer provide nonlinearity to the ELM. Over the past decade, a lot of extensive research has been done on ELM for several purposes: higher performance, less training time and manual intervention [50], and training on large scale datasets [46]. However, there are still some questions that need further study. What is the relationship between variables and ELM performance? Does random selection of hidden neuron weights and bias provide optimal performance? To what extent can ELM be applied with large datasets? Does ELM training achieve the best training time? Can the ELM performance be guaranteed for any dataset?

The ELM algorithm was invented in 2004 by Huang et al. [55] for training SLFNs and nowadays it is a popular neural network architecture [14]. This network consists of three layers: input, hidden and output layers. Fig 1.1 represents a network of this kind[1]. Symbols **x** and **v** are the input and output vectors, a_{hi} and b_{ch} represent the input and output weights, **y** is the activation (or output) vector of the hidden layer, d_h and f_c are the bias for hidden and output s, and $g(x)$ is the activation function. The SLFN training pursuits to decide the weights and biases of the connections between input and hidden layers (input weights), and between hidden and output layers (output weights) in order to reach the optimal performance in prediction for a given dataset. Liang [74] showed the ability of the SLFN to randomly fix

[1]Henceforth, lowercase bold symbols denote vectors and uppercase bold symbols denote matrices.

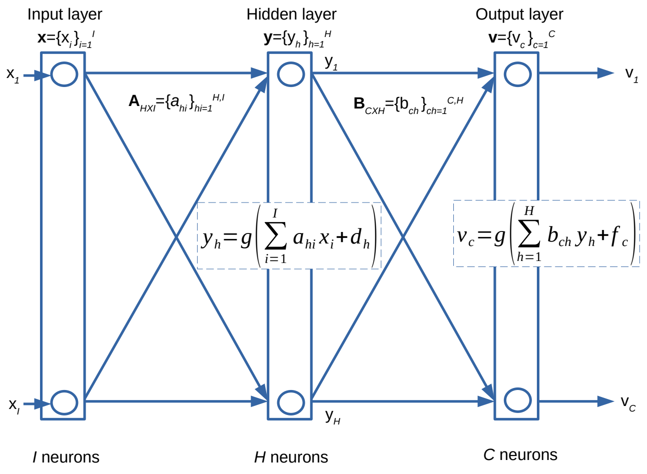


Figure 1.1: Diagram of a SLFN. See text for details.

the input weights. The same goes for the output weights. However, the algorithm could not adjust the weights simultaneously since there was no gain provided. The ELM was proposed to cope with this limitation.

In order to explain the operation of a ELM, let us consider a classification problem with C classes defined by the training set $\{\mathbf{x}_n, \mathbf{t}_n\}_{n=1}^N$, with N patterns $\mathbf{x}_n$ of dimension I, being $\mathbf{t}_n = \{t_{cn}\}_{c=1}^C$ so that $t_{cn} = 1$ only when c is the class label of $\mathbf{x}_n$, and $t_{cn} = 0$ otherwise. Let us consider a ELM network with I input neurons, H hidden and C output neurons, such as in Fig. 1.1. Let $\{a_{hi}\}_{hi=1}^{H,I}$ be the input weights, where a_{hi} connects the i-th input neuron and the h-th hidden neuron, with $h = 1,\dots,H$ and $i = 1,\dots,I$. Let also $\{d_h\}_{h=1}^H$ be the offsets of the H hidden neurons, and $\{b_{ch}\}_{ch=1}^{C,H}$ be the output weights connecting the h-th hidden neuron and the c-th output neuron. The ELM training randomly fixes the inputs weights a_{hi} and biases d_h, and analytically determines the output weights b_{ch}. For the training pattern $\mathbf{x}_n$, the output function $f_c(\mathbf{x}_n)$ of the c-th output neuron, with $c = 1,\dots,C$, is given by:

$$f_c(\mathbf{x}_n) = \sum_{h=1}^{H} b_{ch} g(\mathbf{a}_h^T \mathbf{x}_n + d_h) \qquad (1.1)$$

where b_{ch} is the output weight connecting the h-th hidden and the c-th output neurons, $g(\cdot)$

is the activation function (see below), and $\mathbf{a}_h = \{a_{hi}\}_{i=1}^{I}$ is the weight vector connecting the input layer and the h-th hidden neuron. For N training patterns $\{\mathbf{x}_n, \mathbf{t}_n\}_{n=1}^{N} \in \mathrm{I\!R}^I \times \mathrm{I\!R}^C$, it has been proven that when $H \to N$, the ELM can approximate the output with zero error, so that:

$$\sum_{n=1}^{N}\sum_{c=1}^{C} |f_c(\mathbf{x}_n) - t_{cn}| = 0 \tag{1.2}$$

i.e., there exist $\{\mathbf{a}_h, d_h\}_{h=1}^{H}$ and $\{b_{ch}\}_{ch=1}^{C,H}$ such that:

$$\sum_{h=1}^{H} b_{ch} g(\mathbf{a}_h^T \mathbf{x}_n + d_h) = t_{cn}, \quad n = 1, \ldots, N; c = 1, \ldots, C \tag{1.3}$$

Denoting $y_{hn} = g(\mathbf{a}_h^T \mathbf{x}_n + d_h)$, the previous equation remains:

$$\sum_{h=1}^{H} b_{ch} y_{hn} = t_{cn}, \quad n = 1, \ldots, N; c = 1, \ldots, C \tag{1.4}$$

The equality happens when $H = N$, i.e., when there are so many hidden neurons as training patterns), in which case $\mathbf{Y}$ is square of order N and the number of equations (CN) equals the number of unknowns (CH values of b_{ch}), so the previous system of equations becomes definite. On the contrary, when $H < N$ the number of unknowns is lower than the number of equations, so the system is over-determined, i.e., there are infinite solutions. Since N (number of training patterns) may be high in large-scale datasets, while H can not be large in order to avoid an excesively slow network training, this case will be the most frequent situation. Therefore, let us seek for an approximated solution for $H < N$. Defining $\mathbf{B} = \{b_{ch}\}_{ch=1}^{C,H}$, a $C \times H$-order matrix; $\mathbf{T} = \{t_{cn}\}_{cn=1}^{C,N}$, a $C \times N$-order matrix; and $\mathbf{Y} = \{y_{hn}\}_{hn=1}^{H,N}$, a $H \times N$-order matrix; the CN eqs. 1.4 can be written compactly as:

$$\mathbf{BY} = \mathbf{T} \tag{1.5}$$

Note that when $H = N$, if the $\mathbf{Y}$ matrix is full-range (i.e., $\det(\mathbf{Y}) \neq 0$ and exists $\mathbf{Y}^{-1}$) the output weights can be calculated as $\mathbf{B} = \mathbf{TY}^{-1}$, and in this case the solution is unique. However, when $H < N$ the order of matrix $\mathbf{Y}$ is $N \times H$, so it is non-squared and $\mathbf{Y}^{-1}$ does not exist. However, its Moore-Penrose pseudo-inverse [100]$\mathbf{Y}^{\dagger}$ does exist and is of order $H \times N$. Thus, the output weight matrix $\mathbf{B}$ can be calculated as:

$$\mathbf{B} = \mathbf{TY}^{\dagger} \tag{1.6}$$

so the output weights $\{b_{ch}\}_{ch=1}^{C,H}$ are calculated in a closed-form using a simple matrix product and matrix pseudo-inversion. The ELM training method is compiled by algorithm 1. The singular value decomposition (SVD) method is often used to calculate the Moore–Penrose generalized inverse $\mathbf{Y}^\dagger$ in most of the ELM implementations.

Algorithm 1: Extreme learning machine for classification, version 1.

```
1 Algorithm: [$\mathbf{A},\mathbf{B}$]=ELM1($\mathbf{X}$,$\mathbf{T}$,$H$,$g$)
    Data: $\mathbf{X}=\{x_{ni}\}_{ni=1}^{N,I}$: training set; $\mathbf{T}=\{t_{cn}\}_{cn=1}^{C,N}$: true output; $H$: number of hidden
          neurons; $g$: activation function
    Result: $\mathbf{A}=\{a_{hi}\}_{hi=1}^{H,I}$: input weights; $\mathbf{d}=\{d_h\}_{h=1}^{H}$: bias; $\mathbf{B}=\{b_{ch}\}_{ch=1}^{C,H}$: output
            weights.
2 Randomly intialize $\mathbf{A}$ and $\mathbf{d}$.
3 Calculate hidden layer activity: $\mathbf{Y}=\{y_{hn}\}_{hn=1}^{H,N} \leftarrow \{g(\mathbf{a}_h^T\mathbf{x}_n+d_h)\}_{hn=1}^{H,N}$.
4 Output weight: $\mathbf{B} \leftarrow \mathbf{T}\mathbf{Y}^\dagger$.
```

The input data $\{\mathbf{x}_n\}_{n=1}^{N}$ must be scaled the range [-1,1]. The activation function $g(x)$ is a critical part because it defines the range of hidden neuron outputs. Several functions are commonly used (see eqs. 1.7-1.12): sigmoid, sinusoid, hard limit, triangular basis, radial basis and cosinusoid.

$$g(x)=\frac{1}{1+\mathrm{e}^{-x}} \tag{1.7}$$

$$g(x)=\sin(x) \tag{1.8}$$

$$g(x)=\begin{cases}1 & x\geq 0\\ 0 & x<0\end{cases} \tag{1.9}$$

$$g(x)=\begin{cases}1-|x| & |x|\geq 1\\ 0 & |x|<1\end{cases} \tag{1.10}$$

$$g(x)=\begin{cases}1-|x| & |x|\leq 1\\ 0 & |x|>1\end{cases} \tag{1.11}$$

$$g(x)=\cos(x) \tag{1.12}$$

Thus, the ELM is a neural network whose input weights and biases are randomly generated, and whose output weights are determined analytically using a closed-form expression [9] through the product of the generalized inverse of the hidden layer activity matrix $\mathbf{Y}$ and the true output matrix **T** [57]. The ELM has been proved to be a universal approximator, that for $H \to N$ tends to learn correctly, i.e. with zero error, the training set [72, 117]. Besides, these networks provide an integrated learning platform with a widespread feature mapping and can be applied directly to classification and regression problems [51, 56].

1.4 Variants of the extreme learning machine

Many algorithms have been used in the last decade to improve ELM performance for real applications. In this part of the thesis, we will discuss the latest advanced ELM variants. The randomness of the hidden neurons enables ELM to train quickly, but it leads to fluctuation in the performance of the classification problem, and calculating the output weights is the last step in this algorithm. Starting from these two points, the researchers devoted their efforts to improve the ELM robustness and stability.

One of the iterative methods for ELM is to incrementally add hidden neurons [54] (incremental ELM, I-ELM), that was extended using random search to avoid non-significant neurons [53]. However, this method is time-consuming, and was improved [52] by recalculating the output weights of the existing neurons based on a convex optimization method when a new hidden neuron is added. The incremental regularized extreme learning machine (IR-ELM) adds hidden neurons one by one, updating recursively the output weights in an efficient way [120]. The incremental learning schemes allow ELM to automatically adjust the number of hidden neurons to obtain a better performance for regression and classification compared to random initialization, but they are more time consuming, that is undesirable for real applications. The twin ELM [113] is a version proposed for classification that uses two non-parallel hyperplanes that simultaneously minimize the distance to one class while keeps away from the other class. The online sequential ELM (OS-ELM) allows to learn pattern-by-pattern or chunk-by-chunk [75] and has also been combined in ensembles [68]. In [112], an adaptive ensemble of ELM networks is applied to the prediction of non-stationary time series. In [132], the augmented OS-ELM is used for classification and regression of noncircular quaternion signals, that provide a convenient way to represent 3D and 4D signals. Other ensembles of ELM have been proposed in [92], using negative correlation learning, and [82], a committee of voting ELMs trained with different bootstrap training samples for road lane landmark detection. Low rank matrix factorization is also used in the ELM autoencoder [91], that learns optimal low dimensional features for the same application. The weighted ELM [135] is oriented to classification of imbalanced data, that has been applied in [49] for discriminative data clustering alongside with linear discriminant analysis and K-means, a clustering method that was also used with ELM in [80] to forecast sales of computer servers. The constrained ELM [133, 134] for classification selects the input weights randomly replacing a collection of arbitrary random values, as in ELM, by a set of differences between class samples.

The ELM has been combined with other paradigms like fuzzy logic, used in [41] to find the optimal ELM hyper-parameters (size of the hidden layer), and particle swarm optimization (PSO) for feature selection and hidden layer size estimation in the sleep stage classification over electrocardiogram signal [110]. The ELM and PSO are also combined in [69] with boosting for electric consumption time series forecasting. The evolutionary ELM [131] uses a differential evolutionary algorithm to set the input weights instead of random weight initialization in order to achieve compact networks. In [63] the ELM is applied instead of genetic algorithms for symbolic regression in system identification. Deep neural networks have also been combined with ELM by using the recursive deep arc-cosine kernel [2]. Regarding hardware implementations, [87] describes a neuromemristive circuit architecture for ELM, and [71] compares experimentally several implementations on different hardware devices.

Paper [50] compared ELM and support vector machine (SVM) and found out that ELM was equivalent to SVM in classification problems, being more probably that ELM reach better generalization performance. The ELM was also extensively compared to SVM in paper [78], and to SVM and random forest in [3]. In paper [58], ELM has milder optimization constraints compared to least-square support vector machine (LS-SVM) and proximal support vector machine (PSVM), yielding better classification performance with fewer optimization constraints. Bayesian ELM outperformed the classical ELM in six different regression tasks [108]. There exist also some ELM defficiencies identified by researchers, such as difficulties to estimate the oscillation bounds on the ELM generalization performance [17, 31].

1.5 Large scale datasets and big data

The ELM has great performance in a wide range of problems, but it is not specially suitable for dealing with large scale datasets because it requires very intensive computation, is time-consuming, and may lead to out-of-memory problems. However, some studies have a completely different opinion, claiming that ELM is a perfect choice for solving big data problems [126]. The literature reports several works that use ELM with large scale datasets [16].

The study [114] uses the bag of little bootstraps technique to reduce the size of the training datasets and alleviate the computational overhead of a bagging ensemble of ELMs for large-scale datasets. In [119] the symmetric ELM cluster is evaluated for traffic congestion prediction, transforming a large-scale problem (up to 5 millions of training patterns) in several sub-problems of small and medium size datasets. In [35], feature selection is used to reduce the data size (up to 58,000 patterns and 60 features), identifying the most relevant features by ranking them with the coefficient obtained through ELM divided by the variation coefficient. The regularized ELM [60] was applied to large-scale image classification problems up to 250,000 training patterns, 1,770 inputs and 340 classes, spending 65 s. for training on a 4-core computer with CPU i7–3630, 2.4GHz and 8GB RAM in 2015. In [61], the approximated kernel ELM was evaluated on middle-sized datasets (up to 14,000 training patterns). Paper [77] refers that ELM cannot handle effectively high-dimensional data because its generalization performance tends to become worse in these cases. In fact, the original ELM cannot handle big high dimensional data [48], [127], it requires more hidden neurons than conventional SLFN tuning algorithms, increases the network size and cannot be parallelized due to the dependence of pseudo-inverse calculation on SVD [90].

The fast singular value decomposition (SVD)-hidden-neurons based ELM (FSVD-H-ELM) removes the random weight initialization by applying SVD to multiple random subsets sampled from the original dataset [28]. However, SVD is much slower than random generation, and requires more memory than pseudo-inverse calculation, so it can only be applied to small matrices. Finally, the matrix pseudo-inverse continues to be calculated, so globally FSVD-H-ELM is more complex than ELM. The overload introduced by SVD is only compensated by the random selection of training patterns, that keeps small the matrices to which SVD is applied. This work uses a fast divide-and-conquer scheme to keep tractable the computational complexity on high volume data, up to 20 million patterns and 30 million inputs (dataset `KDD2010`), spending 1,769 seconds per fold (about 30 minutes) on a poweful computer with Intel Xeon E5-2650 2 GHz CPU (32 cores), 256GB RAM and Matlab R2013a in 2016. However, in dataset `Webspam`, with only 350,000 patterns and 16 million inputs, the training time is 8,154 s./fold (about 2.5 hours), so the time does not only depend on the dataset size, but also on other properties of the dataset.

The review [46] reports recent applications of ELM in the big data context. The papers [42] and [33] describe parallel implementations of ELM based on MapReduce and Spark, respectively, the latter being able to classify 38 millions of patterns with 8 inputs on a computer cluster of 10-35 nodes. However, the use of big data tools such as MapReduce is not effective for ELM because the most time-consuming part is matrix multiplication and pseudo-inversion. The elastic ELM [118] uses MapReduce to speed up the matrix products through incremental, decremental and correctional calculations, being applied to synthetic datasets until 10 millions of patterns but only 50 inputs. Another MapReduce-based approach [125] uses an ELM ensemble for large-scale imbalanced classification datasets with two classes up to 300,000 patterns and class imbalance ratios up to 2,000. The work [5] presents a complete ELM toolbox for big data applications on Matlab and Python using GPU acceleration based on MapReduce, giving a fresh view about ELM compared to traditional linear algebraic performance. An ELM trained with 19,000 hidden nodes on this toolbox spent 15 days and 5 hours in year 2015 to process a dataset of 500 million patterns with 147 inputs, on a powerful workstation of 4-core 4GHz CPU with 256-512GB RAM per core and specific hardware (GPU acceleration). In [81], a ELM with rank-reduced matrix is proposed to decrease the size of the hidden layer, speeding the training while raising the performance on datasets up to 29,000 training patterns. The E-OS-ELM [124] avoids the oscillations of OS-ELM in different trials spending about 1 hour to process large-scale datasets until 4.8 million patterns and 54 inputs. The feature-bagged ELM [62] is another variant that uses a combination of ensembles and feature bagging to run over computer systems with severe memory constraints.

1.6 Applications of ELM networks

There is a wide range of applications where ELM was used giving suitable solutions to machine learning problems, that are often the best, with high speed and generalization ability. It has obtained many honorable achievements over the past period, such as virtual personal assistants, predictions while commuting, social media services, spam email, malware filtering, autonomous cars, driving, speech recognition and medical imaging [10]. The study [11] categorizes the ELM applications as in Fig 1.2:

ELM has been employed in many real applications such as e-healthcare systems, chemistry, economics, image processing, Internet of Things, and robotics, among others. The collaborative ELM with a confidence interval [97] is an enhanced ELM version that eliminates redundant calculations of the network neurons in e-healthcare institutions. The extensive experimental analysis shows that the model is efficient and achieves high accuracy (up to 98%) in diagnosing clinical events by analyzing patients' medical records. Surantha *et al.* [110]

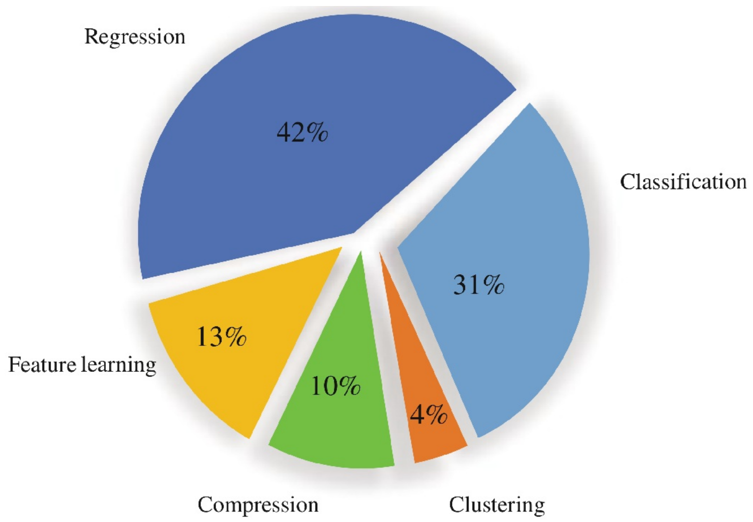


Figure 1.2: Distribution of main application areas of ELM according to [11].

develops an accurate model for classifying sleep stages by features of heart rate variability extracted from electrocardiogram by integrating ELM and particle swarm optimization for selecting features and determining the number of hidden neurons. The weighted ELM has been applied in chemistry [70] to predict protein-protein interactions using a combination of scale-invariant feature transformations that improves performance compared to SVM. In economics, the work [104] uses ELM for credit risk assess, predicting possible loan defaulters for credit lending institutions in order to avoid further losses. Other applications include: food safety [38, 39]; geography and mapping [47, 89]; Internet of Things [73, 129]; robotics [32, 102]; transportation [76, 115]; recognition of facial expressions [25]; image classification [26, 93]; taste recognition [128]; video anomaly detection [116]; bacterial foraging optimization [21]; complex chemical processes [43] and identification of COVID-19 [95].

CHAPTER 2

QUICK EXTREME LEARNING MACHINE FOR LARGE-SCALE CLASSIFICATION

The ELM has been applied to several tasks such as classification, regression and time series prediction, among others. Classification is one of the main problems to which ELM has been applied, and in this chapter we will focus on it. As commented in chapter 1, the efficiency of ELM is limited to small and medium-sized datasets, because with large datasets several major issues affect to its speed and memory requirements. One of them is related to the number of hidden neurons, that will be denoted as H (see Table 2.2 below for a whole listing of the nomenclature symbols and their meanings). The input weights of ELM, connecting the input and hidden layers, are set randomly, while the output weights are calculated by multiplying the target matrix (class labels in classification problems) by the pseudo-inverse of the matrix with the activations of the hidden neurons. The number H of hidden neurons is relevant because the more training patterns, the more hidden neurons are required to achieve a good performance. This number is a hyper-parameter that must be tuned in order to achieve a good performance. Often, its value is selected from a collection of pre-defined values as the one with the best ELM performance (grid-search approach), although alternative strategies such as fuzzy logic [41] and particle swarm optimization [110] have been also proposed to select the optimal hidden layer size. All these approaches exhibit a high complexity that limits its application to small and medium-sized datasets, since the ELM must be trained and tested many times. One of the purposes of the current research is to propose a method that automatically selects the network size from the dataset without repetitions of the ELM

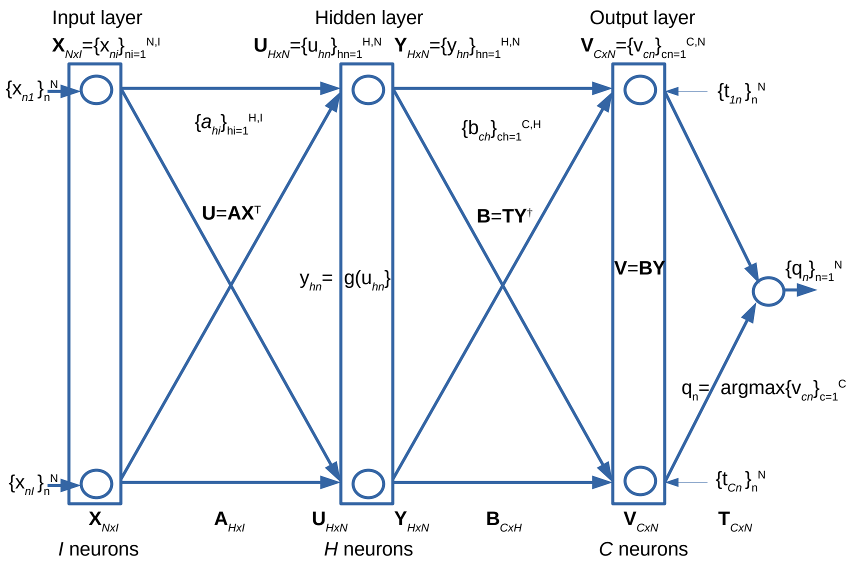


Figure 2.1: Diagram of the ELM neural network for classification during training. During test, $\mathbf{T}_{C\times N}$ is removed, while $\mathbf{X}_{N\times I}$, $\mathbf{V}_{C\times N}$ and q_n are replaced by $\mathbf{S}_{P\times I}$, $\mathbf{V}_{C\times P}$ and z_p, respectively.

training in order to apply the network to large-sized datasets in an efficient manner.

Another issue with the efficiency of ELM for large-scale datasets and big data applications is the calculation of the pseudo-inverse of the hidden layer activation matrix, that may be very slow for large-scale datasets and eventually may not fit in the available memory. Although in the previous chapter we saw several papers that applied ELM for these kind of problems, in some cases the datasets are not so large, and in others the gain on speed is mainly based on the use of big data technologies (MapReduce and Spark) or specific hardware (parallelization, GPU acceleration and powerful computing capabilities), that are thus required for an efficient processing. In the current chapter, the objective of the research is to proposes a ELM-based method that can be executed on classification datasets with arbitrarily large populations over general-purpose computers without any specific hardware nor software, such as big data technologies.

Our proposal, called **quick extreme learning machine** (QELM), is designed to allow the execution of ELM on large datasets while keeping its simplicity and speed [12]. Section 2.1

Symbol	Meaning
I	No. inputs
C	No. classes
H	No. hidden neurons
N	No. training patterns
P	No. test patterns
$Q = N + P$	No. total patterns
	Training
$\mathbf{x}_n$	n-th training pattern
$\mathbf{A}_{H\times I} = \{a_{hi}\}_{hi=1}^{H,I}$	Input weights
$\mathbf{U}_{H\times N} = \{u_{hn}\}_{hn=1}^{H,N}$	Input to h-th hidden neuron
$\mathbf{Y}_{H\times N} = \{y_{hn}\}_{hn=1}^{H,N}$	Activity (or output) of h-th hidden neuron
$\mathbf{B}_{C\times H} = \{b_{ch}\}_{ch=1}^{C,H}$	Output weights
$\mathbf{V}_{C\times N} = \{v_{cn}\}_{cn=1}^{C,N}$	Outputs of the output neurons for training patterns
$\mathbf{T}_{C\times N} = \{t_{cn}\}_{cn=1}^{C,N}$	True outputs for the output neurons
$q_n = \arg\max_{c=1,\dots,C}\{v_{cn}\}$	Predicted class label for $\mathbf{x}_n$
$w_n = \arg\max_{c=1,\dots,C}\{t_{cn}\}$	True class label of $\mathbf{x}_n$
	Test
$\mathbf{s}_p$	p-th test pattern
$\mathbf{V}_{C\times P} = \{v_{cp}\}_{cp=1}^{C,P}$	Outputs of the output neurons for test patterns
$z_p = \arg\max_{c=1,\dots,C}\{v_{cp}\}$	Predicted class label for $\mathbf{s}_p$

Figure 2.2: List of symbols used in the text and their meaning.

describes ELM network, and section 2.2 discusses the role of the number H of hidden neurons. The method proposed for the estimation of the hidden layer size in QELM is described in section 2.3, while section 2.4 explains the output weight calculation and section 2.5 compiles the whole algorithm. Results are reported and discussed in section 2.6.

2.1 Extreme learning machine

First we introduce the notation associated to the ELM network (refer to Table 2.2 and to Figure 2.1). Let Q be the total number of patterns, being N and P the numbers of training and test patters, respectively, so that $Q = N + P$. Let $\{\mathbf{x}_n\}_{n=1}^{N}$ be the N training patterns of dimension I (number of inputs of the ELM network) composing the matrix $\mathbf{X}$, of order $N \times I$. In this chapter we will not consider the biases d_h separated from the input weights, so the value I will be 1 plus the original dimensionality of the input pattern. Given the number of classes C of the classification problem, let $w_n \in \{1,\ldots,C\}$, with $n = 1,\ldots,N$, be the true class label of $\mathbf{x}_n$, composing the N-dimensional vector $\mathbf{w} = (w_1,\ldots,w_N)$. Let H be the number of neurons in the hidden layer of the ELM, and $\{a_{hi}\}_{hi=1}^{H,I}$ the weights of the hidden neurons (input weights)[1], composing the matrix $\mathbf{A}_{H\times I}$. Note again that this matrix also includes the biases d_h of the classical ELM. Let $\{u_{hn}\}_{hn=1}^{H,N}$ be the inputs of the hidden neurons, composing the matrix $\mathbf{U}_{H\times N}$, and $\{y_{hn}\}_{hn=1}^{H,N}$ their activations (or outputs), composing the matrix $\mathbf{Y}_{H\times N}$. Let $\{b_{ch}\}_{ch=1}^{C,H}$ be the weights conecting the hidden and output layer (output weights), composing the matrix $\mathbf{B}_{C\times H}$, where the output layer of the ELM has C neurons, one for each class. The desired outputs for the output neurons are $\{t_{cn}\}_{cn=1}^{C,N}$ given by $t_{cn} = \delta(c,w_n) = 1$, where $\delta(c,w_n) = 1$ for $c = w_n$ and $\delta(c,w_n) = 0$ otherwise, composing the matrix $\mathbf{T}_{C\times N}$. The P test patterns, of dimensionality I, are $\{\mathbf{s}_p\}_{p=1}^{P}$, and compose the matrix $\mathbf{S}_{P\times I}$. Let $\mathbf{V}_{C\times P}$ be the output of the last layer, and $\{z_p\}_{p=1}^{P}$ the class labels predicted by the ELM, for the test patterns $\{\mathbf{s}_p\}_{p=1}^{P}$, composing the P-dimensional vector $\mathbf{z} = (z_1,\ldots,z_P)$.

Let us consider a training pattern $\mathbf{x}_n = (x_{n1},\ldots,x_{nI})$ with inputs $x_{ni} \sim \mathscr{N}(0,1)$, i.e., standarized with zero mean and standard deviation one, with $n = 1,\ldots,N$, and $i = 1,\ldots,I$. The input u_{hn} of a hidden neuron h is $u_{hn} = \sum_{i=1}^{I} a_{hi}x_{ni}$. In matrix form for all the hidden neurons and training patterns, $\mathbf{U} = \mathbf{A}\mathbf{X}^T$, where $\mathbf{X}^T$ denotes the transposed of matrix $\mathbf{X}$. The output of this neuron is given by $y_{hn} = g(u_{hn})$, being $g(u) = (1+\mathrm{e}^{-u})^{-1}$ the activation function[2]. The matrix $\mathbf{A}$ is defined with random numbers, whose values, given the activation function $g(u)$, should verify $|a_{hi}| < 1/I$ in order to guarantee that u_{hn} is restricted to a symmetric range around $u = 0$ for $x_{ni} \sim \mathscr{N}(0,1)$. The output weight matrix $\mathbf{B}_{C\times H}$ is calculated as $\mathbf{B} = \mathbf{T}\mathbf{Y}^\dagger$, where $\mathbf{Y}^\dagger$ is the pseudo-inverse matrix of $\mathbf{Y}$. The trained ELM neural network is defined by the weight matrices $\mathbf{A}$ and $\mathbf{B}$, of sizes $H\times I$ and $C\times H$ respectively, so the number of weights is $H(I+C)$. Since I and C are given by the dataset, the network size is completely determined by the number H of hidden neurons. Algorithm 2 reports the ELM pseudocode for classification.

[1]The notation $\{a_{hi}\}_{hi=1}^{H,I}$ means $h = 1,\ldots,H$ and $i = 1,\ldots,I$.

[2]An exponent $\alpha \neq 1$ may be used in the exponential, and alternative activation functions listed in eq. 1.7–1.12 of chapter 1 may be also used.

Algorithm 2: Extreme learning machine for classification, version 2.

```
1  Algorithm: [A, B, q, z]=ELM2(X,w,H,g,S)
   Data: X: matrix of size $N \times I$ with the $N$ $I$-dimensional train patterns $\{\mathbf{x}_n\}_{n=1}^N$;
         $\mathbf{w} = (w_1, \dots, w_N)$: true class labels $w_n \in \{1, \dots, C\}$ of $\{\mathbf{x}_n\}_{n=1}^N$; $H$: number of
         hidden neurons; S: matrix of size $P \times I$ with the $P$ $I$-dimensional test patterns
         $\{\mathbf{s}_p\}_{p=1}^P$.
   Result: A: matrix with the ELM input weights, of size $H \times I$; B: matrix with the
           ELM outputs weights, of size $C \times H$; q (resp. z): vector of length $N$ (resp. $P$)
           with the predicted class labels for the training (resp. test) patterns.
2  $\mathbf{T} \leftarrow \{\delta(c, w_n)\}_{cn=1}^{C,N}$ ; // $\delta(c, w_n) = 1$ when $c = w_n$ and 0 otherwise
3  $\mathbf{X} \leftarrow [\mathbf{1}, \mathbf{X}]; \mathbf{S} \leftarrow [\mathbf{1}, \mathbf{S}]$ ; // paste column vector $\mathbf{1}$ on the left
4  $\mathbf{A} \leftarrow$ rand($[-1/I, 1/I], H, I$) ; // random values in $(-1/I, 1/I)$
5  $\mathbf{U} \leftarrow \mathbf{A}\mathbf{X}^T$
6  $\mathbf{Y} \leftarrow g(\mathbf{U})$ $\mathbf{B} \leftarrow \mathbf{T}\mathbf{Y}^\dagger$
7  $\mathbf{V} \leftarrow \mathbf{B}\mathbf{Y}$
8  $\mathbf{q} \leftarrow (q_1, \dots, q_N)$; $q_n \leftarrow \arg\max_{c=1,\dots,C}\{v_{cn}\}, n = 1, \dots, N$
9  $\mathbf{V} \leftarrow \mathbf{B}g(\mathbf{A}\mathbf{S}^T)$
10 $\mathbf{z} \leftarrow (z_1, \dots, z_P)$; $z_p \leftarrow \arg\max_{c=1,\dots,C}\{v_{cp}\}, p = 1, \dots, P$
```

2.2 Number of hidden neurons

The number H of hidden neurons is often tuned using the grid-search method, i.e., to use K-fold cross validation with three datasets per fold: training, validation and test sets. For each fold, the ELM is trained using a value of H and its performance is evaluated on the corresponding validation set. The performance for this value of H is averaged over the K folds. The training-test cycle must be repeated for each value of H from a pre-specified collection, and the one with the best average performance is selected for testing. Then, for each fold the ELM using this value of H is trained over the training and validation sets, and tested on the test set. The process is repeated for the K folds, and the final test performance is the average over the K folds.

Alternative methods for model selection in ELM include [123], that uses pruning of the less significant hidden neurons from an initially large network, but this approach is relatively slow, spending about 5 minutes on the largest datasets considered (5,600 patterns and 64 inputs). The paper [30], instead of tuning the network size, uses a large fixed value of H, pruning those neurons that are non-relevant or similar to other non-relevant neurons using simulated annealing. The process is stopped when the desired network size is achieved, without adjusting weights nor reducing performance. The complexity of this approach is high, spending about 300-900 seconds (5-15 minutes) on dataset `adult` with 39,047 training patterns. The method U-ELM [85] adds incrementally neurons to the hidden layer using a PSO-based multi-objective function that minimizes the uncertainty using a Riemann metric in order to select the optimal input weights. The addition of new neurons stops when a maximum number of iterations is reached or uncertainty does not change appreciably. This method is also relatively slow, spending about 20 seconds on a small dataset with 350 training patterns. The study [103] provides a review of methods to set an appropiated H value for the ELM.

2.3 Selection of H in QELM

With large datasets, to use grid-search for hyper-parameter tuning is very slow because the ELM must be executed several times, depending on K (number of folds) and the number of H values tried. The literature (see e.g. Tables 2 and 3 in [123] and Table 2 in [85]) has shown that the optimal H raises with the number N of training patterns. Since H is the number of rows of the hidden activation matrix $\mathbf{Y}$, large datasets with high N require activation matrices with many rows, that slows down the pseudo-inverse calculation. On the other hand, this calculation requires to store in memory the whole matrix $\mathbf{Y}$. However, increasing N, and consequently H, the activation matrix will eventually not fit in memory. Our proposal is to calculate H as an increasing function of N with an upper bound to keep limited the number of rows or matrix $\mathbf{Y}$ for large datasets. In order to estimate this function, we evaluated the behavior of the ELM performance, measured by the Cohen kappa statistic [19], with H over the small datasets of our collection (see subsection 2.6) varying H from $H = 1$ to $H = N$ (100 values). The input weights $\{a_{hi}\}_{hi=1}^{H,I}$ are set to random values, so the classification performance has a certain degree of randomness. In order to reduce this randomness, we generated a random weight matrix $\mathbf{A}_0$ of size $N^* \times I$, being N^* the largest N over the small datasets. For each value of H, the matrix $\mathbf{A}$ uses only the first H rows of $\mathbf{A}_0$, with H ranging from $H = 1$ to $H = N$, with $N \leq N^*$. In this way, the variability due to randomness is almost removed.

Figure 2.3 plots the best values of H against the number of training patterns and inputs. The left panel reports a increasing dependence between H and N, in fact their correlation

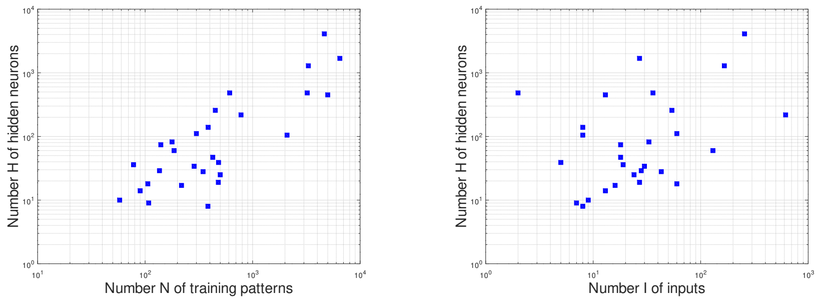


Figure 2.3: Value of H (in logarithmic scale) that achieved the best kappa over the small datasets (Q <15,000 patterns) of the collection described in section 2.6 vs. the number N of training patterns (left panel) and vs. the number I of inputs (right panel), both in logarithmic scales.

coefficient is 0.71, a value that can be considered "from moderate to good" according to the Colton scale [22]. However, the right panel does not reflect a clear dependence of H on I, with a "bad to moderate" correlation (0.32). Therefore, the best value for H seems to depend more on N than on I, being increasing with N. Figure 2.4 plots the kappa score for three datasets with slightly different behaviors for the performance in the upper panels and lower left panel, and the average over a wide dataset collection (see subsection 2.6) in the lower right panel. In the four panels, kappa is plotted for $0 < H/N \leq 1$. The training performance raises with H/N until perfect classification, but the test kappa raises only for H/N below 0.2 and decreases or keeps relatively constant for larger H/N, suggesting that the highest performance is expected for $0.1 \leq H/N \leq 0.2$. We also saw in our experiments that kappa reduced even more for $H > N$. Therefore, we propose that QELM uses $H = \lfloor \eta N \rfloor$ with η=0.15, because in Figure 2.4 (lower right panel) the highest average test performance is achieved for $H/N = \eta \sim 0.15$. Since H must be upper bounded, we set a threshold N_0 for N so that $H = \lfloor \eta N \rfloor$ for $N < N_0$ (datasets that can be considered of small size) and $H = \lfloor \eta N_0 \rfloor$ for $N > N_0$ (large datasets). We set $N_0 = 15,000$, so the maximum H is $\eta N_0 = 0.15 \cdot 15,000 = 2,250$ hidden neurons. Thus, H is given by:

$$H = \lfloor \eta \min(N, N_0) \rfloor, \quad \eta = 0.15, N_0 = 15{,}000 \tag{2.1}$$

This approach allows to estimate a value for H directly from the number N of training patterns, which is an intrinsic property of the dataset, without the need to execute the ELM

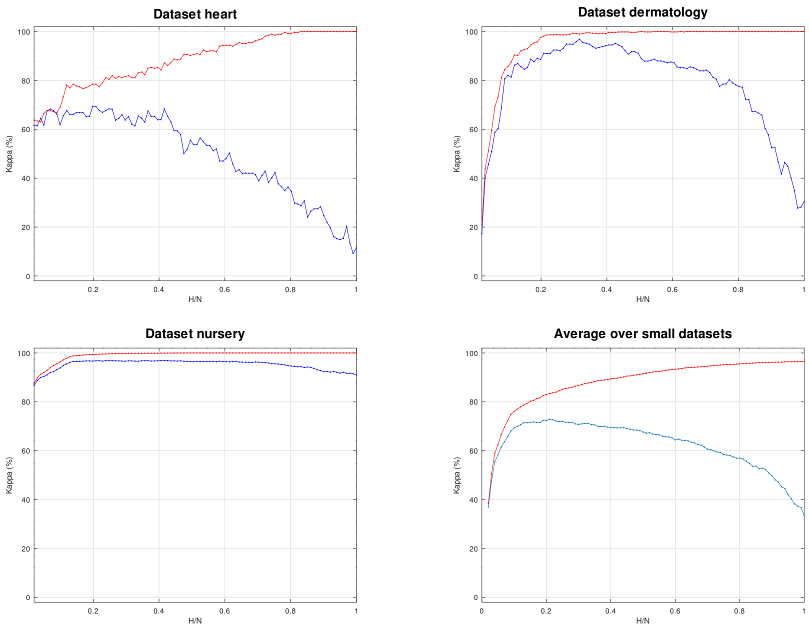


Figure 2.4: Training (continuous red line) and test (dashed blue line) Cohen kappa (19) vs. H/N for three datasets that are illustrative of the behavior of the ELM performance depending on H/N (upper panels and lower left panel). The lower right panel plots the average kappa over the small datasets (Q <15,000 patterns) of the collection described in section 2.6.

training or test stages, nor to perform complex calculations over the original training set, that might be slow for large datasets. The value of H proposed by eq. 2.1 is proportional to N for small datasets with $N < N_0$=15,000 patterns, reaching its highest value H=2,250 for N_0=15,000 patterns and remaining constant for large datasets with higher N. The proposed value for H, that is the number of rows in matrices $\mathbf{Y}$ and $\mathbf{A}$, and the number of columns of $\mathbf{B}$, does not raise for large N, so that the time spent by the ELM training raises slowly with the dataset size. Besides, the value of H in eq. 2.1 is not expected to be very far from optimality, because the ELM performance (see Figure 2.4) achieves its maximum for low values (below 0.2) of H/N. Finally, larger values of H with respect to N, e.g. H =0.9N, are not required because our experiments report that: 1) performance keeps constant or reduces for $H > N$ or H >0.4N, see lower right panel of Figure 2.4; and 2) the ELM training, and specifically the matrix storage and pseudo-inversion, becomes too slow or even not possible for H >15,000 hidden neurons and medium-size datasets.

2.4 Calculation of the output weights in QELM

The output weights of an ELM network are calculated using the pseudo-inverse of matrix $\mathbf{Y}$, of order $H \times N$. This is an expensive task either in terms of memory, because it requires to store a matrix that may be large, and time, due to the computational complexity of the matrix pseudo-inversion, that is similar to matrix inversion. Specifically, the complexity of the n-order square matrix inversion [64] is $\mathcal{O}(n^3)$ using the standard Gauss-Jordan elimination, $\mathcal{O}(n^{2.807})$ using the Strassen algorithm [109], $\mathcal{O}(n^{2.376})$ using the Coppersmith–Winograd algorithm [23], used e.g. in manifold learning ELM (ML-ELM) for matrix inversion [84], and $\mathcal{O}(n^{2.373})$ using methods inspired on Coppersmith–Winograd. Considering specifically the matrix pseudo-inversion, given a matrix $\mathbf{G}$ of order $m \times n$ with $m \geq n$, the method `geinv` proposed in [24] uses a full rank Cholesky factorization, followed by the inversion of a symmetric matrix, to calculate the pseudo-inverse of $\mathbf{G}$. The complexities of the factorization and inversion methods proposed in the previous paper are of orders $\mathcal{O}(n^3)$ and $\mathcal{O}(r)$, respectively, being r the rank of $\mathbf{G}^T\mathbf{G}$, but on parallel architectures with enough processors these complexities can be reduced to $\mathcal{O}(n)$ and $\mathcal{O}(\log r)$, respectively. However, even these optimized approaches are very slow to calculate the pseudo-inverse of matrix $\mathbf{Y}$, of size $H \times N$, for large datasets with high N, and consequently high H (numbers of columns and rows of $\mathbf{Y}$, respectively). The previous section proposed to keep H bounded with N. Analogously, the number M of columns of $\mathbf{Y}$ must also be bounded when the size of the training dataset raises, so for large datasets it must be $M \leq N$ instead of $M = N$ as in ELM. Note that an acceptable value for M will be conditioned by N, but not by the number I of inputs. Our proposal is to set an upper bound $M_0 \leq N$ on M so that $M \leq M_0$. For small datasets $M_0 = N$, but $M_0 < N$ for large datasets, so that a smaller training set is used instead the original N training patterns. In order to set M_0, a threshold $N_1 = 5{,}000$ patterns is set on N to separate both cases (small and large datasets). To achieve a reasonable size for matrix $\mathbf{Y}$, the value M_0, that defines its maximum number of columns, is set to:

$$M_0 = \min(N, N_1), \quad N_1 = 5{,}000 \tag{2.2}$$

Specifically, we propose to replace the N original training patterns that ELM uses as columns in $\mathbf{Y}$ by M prototypes created from the original training patterns [4]. A set of prototypes $\{\mathbf{p}_{cl}\}$ of the different classes is defined, with $c = 1,\dots,C$, and $l = 1,\dots,l_c$, being l_c the number of prototypes of class c. Each column of matrix $\mathbf{Y}$ is a prototype, so the number M of columns of matrix $\mathbf{Y}$, that must be lower than M_0, is given by:

$$M = \sum_{c=1}^{C} l_c \leq M_0 \tag{2.3}$$

In datasets where $N < N_1$, we have $M_0 = N$ and $M = M_0$, so all the training patterns can be stored by columns in $\mathbf{Y}$ as in ELM. In datasets where $N > N_1$, we have $M_0 = N_1$ by eq. 2.2, and an upper bound L_c on the number l_c of prototypes of each class is set, so that $l_c \leq L_c$. This bound L_c is class-dependent because classes may be unbalanced, so the most populated classes should have L_c higher than the low populated ones. This suggests that $L_c \sim N_c M_0/N$, where N_c is the number of training patterns of class c. Note that for large datasets, $N > N_1$ and $M_0 = N_1 < N$ so that it may be $N_c M_0/N < 1$ and class c would have zero prototypes. In order to avoid this, a lower bound L_1 is set for L_c, so that a class always has prototypes. To guarantee an minimum number of prototypes per class for large datasets, an acceptable value for the maximum number of prototypes per class would be L_1 =100. However, in datasets with high C where $N_c > L_c$ for all the classes, the number M of prototypes would be $M = L_1 C$. The constraint $M \leq M_0$ requires $L_1 \leq M_0/C$. When $C > M_0$ we have $M_0/C < 1$, so L_1 would be zero and classes would have no prototypes. In order to avoid this posibility, we propose to use:

$$L_1 = \min\left\{L_0, \max\left(1, \left\lfloor \frac{M_0}{C} \right\rfloor\right)\right\}, \quad L_0 = 100 \tag{2.4}$$

so that $L_1 = L_0 = 100$ unless $M_0/C < L_0$, or equivalently $C > M_0/L_0$, in which case L_1 is 1 if $\lfloor M_0/C \rfloor = 0$ and $\lfloor M_0/C \rfloor$ otherwise. Finally, L_c is given by:

$$L_c = \max\left(L_1, \left\lfloor \frac{N_c M_0}{N} \right\rfloor\right) \tag{2.5}$$

Since $l_c \leq L_c$, the classes with $N_c < L_c$ training patterns will have $l_c = N_c < L_c$ prototypes, while classes with $N_c \geq L_c$ will have $l_c = L_c$ prototypes. These prototypes $\mathbf{p}_{cl}$, with $c = 1,\dots,C$, and $l = 1,\dots,l_c$, are calculated in a simple way in order to be efficient for large datasets. The first L_c training patterns of class c are selected as initial prototypes $\{\mathbf{p}_{cl}\}_{l=1}^{L_c}$.

Algorithm 3: Quick extreme learning machine (QELM).

```
1  Algorithm: $[\mathbf{A},\mathbf{B},\mathbf{q},\mathbf{z}]$=QELM($\mathbf{X},\mathbf{w},g,\mathbf{S}$)
   Data: $\mathbf{X}$: matrix with the train patterns, of size $N\times I$; $\mathbf{w}=(w_1,\dots,w_N)$: true class
         labels $w_n\in\{1,\dots,C\}$ of the train patterns; $\mathbf{S}$: matrix with test patterns, of size
         $P\times I$.
   Result: $\mathbf{A}$: matrix with the QELM input weights, of size $H\times I$; $\mathbf{B}$: matrix with the
           QELM output weights, of size $C\times H$; $\mathbf{q}$: vector of length $N$ with the
           predicted class labels for the training patterns; $\mathbf{z}$: vector of length $P$ with the
           predicted class labels for the test patterns.
2  $N_0\leftarrow$15,000; $\eta\leftarrow$0.15; $N_1\leftarrow$5,000; $L_0\leftarrow 100$ ; // Hyper-parameters
3  $M_0\leftarrow\min(N,N_1)$; $L_1\leftarrow\min\left\{L_0,\max\left(1,\left\lfloor\frac{M_0}{C}\right\rfloor\right)\right\}$
4  $H\leftarrow\lfloor\eta\min(N,N_0)\rfloor$; $\{N_c\leftarrow 0\}_{c=1}^{C}$
5  for $n\leftarrow 1,N$ do
6      $c\leftarrow w_n$; $N_c\leftarrow N_c+1$
7  end
8  $\{L_c\leftarrow\max(L_1,\lfloor N_cM_0/N\rfloor)\}_{c=1}^{C}$; $\{\mathbf{p}_{cl}\leftarrow\mathbf{0};N_{cl}\leftarrow 0\}_{cl=1}^{C,L_c}$; $M\leftarrow 0$
9  for $n\leftarrow 1,N$ do
10     $c\leftarrow w_n$
11     if $l_c<L_c$ then
12         $l\leftarrow l_c$; $\mathbf{p}_{cl}\leftarrow\mathbf{x}_n$; $N_{cl}\leftarrow 1$; $l_c\leftarrow l_c+1$; $M\leftarrow M+1$
13     else
14         $l\leftarrow\underset{j=1\dots,l_c}{\arg\min}\left|\mathbf{p}_{cj}-\mathbf{x}_n\right|$
15         $\mathbf{p}_{cl}\leftarrow\left(1-\frac{1}{N_{cl}}\right)\mathbf{p}_{cl}+\frac{\mathbf{x}_n}{N_{cl}}$; $N_{cl}\leftarrow N_{cl}+1$
16     end
17 end
18 $\mathbf{X}\leftarrow\{\mathbf{p}_{cl}\}_{cl=1}^{C,L_c}$
19 $[\mathbf{A},\mathbf{B},\mathbf{q},\mathbf{z}]\leftarrow$ELM2($\mathbf{X},\mathbf{w},H,g,\mathbf{S}$) ; // ELM() defined in algorithm 2
```

The following training patterns of this class, if they exist, update their nearest prototypes from $\mathbf{p}_{cl}(t)$ to $\mathbf{p}_{cl}(t+1)$ according to:

$$\mathbf{p}_{cl}(t+1) = \left[1 - \frac{1}{N_{cl}(t)}\right]\mathbf{p}_{cl}(t) + \frac{\mathbf{x}_n}{N_{cl}(t)} \tag{2.6}$$

$$c = w_n, \quad l = \underset{j=1,\ldots,l_c}{\arg\min}\left\{|\mathbf{p}_{cj} - \mathbf{x}_n|\right\}, \quad N_{cl}(t+1) = N_{cl}(t) + 1$$

where $\mathbf{p}_{cl}(t+1)$ and $\mathbf{p}_{cl}(t)$ are the new and old versions, respectively, of the l-th prototype of class $c = w_n$, i.e., the class label of training pattern $\mathbf{x}_n$. Besides, $|\mathbf{x}-\mathbf{y}| = \sum_{i=1}^{I}|x_i - y_i|$, being $\mathbf{x}$ and $\mathbf{y}$ two vectors of dimension I. On the other hand, $N_{cl}(t)$ is the number of training patterns of class label c for which prototype $\mathbf{p}_{cl}(t)$ was the nearest one until now. When class c has $N_c < L_c$ training patterns, its l-th prototype $\mathbf{p}_{cl}$ is the l-th training pattern $\mathbf{x}_n$ of class $c = w_n$ for $l = 1, \ldots, N_c$. This updating method, that is an efficient on-line version of the simple K-means clustering algorithm, allows to create a collection of class prototypes in a fast way without excessive memory requirements, because the total number of prototypes $M = \sum_{c=1}^{C} l_c$ for large datasets with $N_cM_0/N > L_1$, using eqs. 2.2 and 2.5, is upper bounded by N_1:

$$\begin{aligned} M &= \sum_{c=1}^{C} l_c \leq \sum_{c=1}^{C} L_c = \sum_{c=1}^{C} \max\left(L_1, \left\lfloor \frac{N_cM_0}{N} \right\rfloor\right) = \\ &= \sum_{c=1}^{C} \left\lfloor \frac{N_cM_0}{N} \right\rfloor \leq \frac{M_0}{N}\sum_{c=1}^{C} N_c = \frac{M_0}{N}N = M_0 \leq N_1 \end{aligned} \tag{2.7}$$

2.5 The QELM algorithm

Algorithm 3 reports the pseudocode of QELM, where the number H of hidden neurons is given by eq. 2.1 and class prototypes are updated according to eq. 2.6. Table 2.1 lists the names, values and meanings of the hyper-parameters of QELM. With these values, $H \leq 2,250$ and $M \leq N_1 = 5,000$, so the size $H \times M$ of the matrix $\mathbf{Y}$ to be pseudo-inverted can not overcome $2,250 \times 5,000$, while matrices $\mathbf{A}$ and $\mathbf{B}$ can not overcome sizes 2,250$\times I$ and $C\times$2,250, being C and I the numbers of classes and inputs, respectively. The values of hyper-parameters N_0, η, N_1 and L_0 are selected in order to limit the computational cost of QELM for large datasets. The value $N_0 = 15,000$ is selected to keep H acceptably low when N is large and to avoid slow training. The value $\eta = 0.15$ is selected to provide a good performance (see Figure 2.4) while keeping H low. Similarly, the value $N_1 = 5,000$ is selected to avoid an

Table 2.1: Hyper-parameters of QELM.

Name	Value	Description
N_0	15,000	Maximum N for which $H = \lfloor \eta N \rfloor$
η	0.15	Fraction of N used for H when $N < N_0$
H	$\lfloor \eta \min(N, N_0) \rfloor$	Number of hidden neurons
N_1	5,000	Maximum number of prototypes (columns of $\mathbf{Y}$)
L_0	100	Lower limit to the number of prototypes of a class when $C < M_0/L_0$

excessive number of prototypes that would slow down QELM in large datasets. Finally, the value $L_0 = 100$ is selected to guarantee a minimum number of prototypes for low populated classes in unbalanced datasets, but not too high in order to avoid an excessive number of prototypes when many classes are present. The performance of QELM is expected to be not very sensitive to these hyper-parameters in order to avoid their tuning, that would hinder its application to large datasets. Therefore, QELM does not require to perform hyper-parameter tuning, e.g. using grid-search, a process that would make this method not practical for large-scale datasets.

The classical ELM has been formally proven to be a universal approximator that can exactly learn any training dataset provided a H large enough is used. However, in practice the ELM can not be applied to datasets with high N because it requires to perform pseudo-inversion of matrix $\mathbf{Y}$ of size $H \times N$. The QELM extends ELM for large datasets with two key contributions. First: it estimates an adequate number H of rows of matrix $\mathbf{Y}$ directly from the dataset properties, without train nor test the network. Our preliminar exploration of the ELM behavior depending on H (see subsection 2.3) reports that the best performance is achieved for low H/N values, specifically 0.1$\leq H/N \leq$0.2. Thus, we propose to use $H = \lfloor \eta N \rfloor$ with η =0.15, bounded by $H = \lfloor \eta N_0 \rfloor$ for large datasets where $N > N_0$. Second: in the calculation of matrix $\mathbf{Y}$, the QELM replaces the original training patterns by a set of class prototypes, whose size is limited for large datasets in order to keep low the number of columns of $\mathbf{Y}$. An efficient on-line version of the well-known K-means clustering algorithm is used to create a representative collection of class prototypes. Due to the K-means properties, this collection is a reduced but significant version of the original training set, that is too large to be used for the ELM training. Combined together, both contributions are oriented to achieve a matrix **Y** of size large enough to achieve a good performance, but small enough to allow its storing in memory and the efficient pseudo-inverse calculation. Since the value estimated for H by QELM is high enough for a good performance on large datasets, and since the class prototypes are guaranteed to be a small but accurate representation of the original training set, the mathematical properties of the ELM guarantee that the classification problem will be learnt with a high level of performance.

Table 2.2: List of small datasets ($Q \leq$15,000 patterns) with the number of total (Q) and training (N) patterns, inputs (I) and classes (C), sorted by increasing N.

No.	Original name	Dataset	Q	N	I	C
1	Breast tissue	tissue	106	58	9	6
2	Hepatitis	hepatitis	155	78	19	2
3	Wine quality	wine	178	90	13	3
4	Sonar, mines vs. rocks	sonar	208	106	60	2
5	Seeds	seeds	210	108	7	3
6	Heart (Statlog)	heart	270	136	28	2
7	Image segmentation	imseg	2,310	140	18	7
8	Ionosphere	ionosphere	351	178	33	2
9	Dermatology	dermatology	366	186	130	6
10	Congressional voting	voting	435	218	16	2
11	Breast cancer Wisc.	wdbc	569	286	30	2
12	Synthetic control chart	synthetic	600	300	60	6
13	Australian credit	australian	690	346	43	2
14	Pima indian diabetes	pima	768	384	8	2
15	Energy efficiency	energy	768	386	8	3
16	Vehicle silhouettes	vehicle	846	426	18	4
17	Annealing	annealing	898	450	54	5
18	Tic-tac-toe	tic	958	480	27	2
19	Mammographic mass	mammograph	961	482	5	2
20	German credit	german	1,000	500	65	2
21	Multiple spirals appart	msa	1,206	612	2	6
22	Isolet	isolet	1,559	780	617	26
23	Abalone	abalone	4,177	2,090	8	3
24	Landsat satellite	sat	6,435	3,222	36	6
25	Musk (v.2)	musk	6,598	3,302	166	2
26	Hand digits USPS	usps	9,298	4,650	256	2
27	Electrical Grid	grid	10,000	5,000	13	2
28	Nursery	nursery	12,958	6,480	27	4

Table 2.3: List of large datasets ($Q > 15{,}000$ patterns) sorted by increasing N.

No.	Original name	Dataset	Q	N	I	C
1	Magic gamma	magic	19,020	9,510	10	2
2	Letter recognition	letter	20,000	10,018	16	26
3	Arabic hand. char.	arabic	16,800	12,600	1,024	28
4	Chess (kr. vs. king)	chess	28,056	14,044	40	18
5	Adult (census income)	adult	48,842	24,422	105	2
6	Poker hand	poker	1,025,010	25,010	10	2
7	Shuttle (Statlog)	shuttle	58,000	43,483	9	7
8	Wisdm smartphone	wisdm	73,803	55,380	92	18
9	Fruits 360 (Kaggle)	fruits	90,380	67,692	30,000	131
10	Devanagari character	devanagari	92,000	69,000	1,024	46
11	IJCNN 2001	ijcnn1	141,691	70,848	22	2
12	Covertype	covtype	581,012	435,768	54	7
13	KDD Cup 1999	kddcup	4,000,000	3,638,724	122	23
14	Record linkage	record	5,749,132	4,311,846	11	2
15	Physical unclonable	physical	6,000,000	5,000,000	128	2
16	Detection IoT botnet	baiot	7,062,606	5,296,869	115	11
17	Hepmass	hepmass	10,500,000	7,000,000	28	2
18	Higgs	higgs	11,000,000	8,249,997	28	2
19	Kitsune network attack	kitsune	21,017,597	8,646,375	115	2
20	Human activ. recog.	human	13,956,557	10,467,372	36	42
21	Heter. activ. recogn.	har	29,097,887	21,823,410	14	6
22	Wesad	wesad	31,470,603	23,602,947	8	4

2.6 Results and discussion

We tested the QELM on a collection of 28 small ($Q \leq 15{,}000$ patterns) and 22 large ($Q > 15{,}000$ patterns) classification problems (see Tables 2.2 and 2.3), including datasets up to 34 million patterns, 30,000 inputs and 131 classes. Most of them are selected from the Machine Learning Repository of the University of California at Irving (UCI)[3]. Dataset `msa` is the well-known artificial dataset with 6 nested spirals, and `fruits` comes from the Kaggle[4] repository. The tables list, for each dataset, its original name, the short name used in this paper, the total

[3]https://archive.ics.uci.edu (Visited May, 2021)

[4]https://www.kaggle.com (Visited May, 2021)

number of patterns (Q) and the number of training patterns (N), inputs (I) and classes (C). On the small datasets, QELM is compared to the classical ELM and the support vector machine with radial basis function (RBF) kernel, henceforth named SVC, implemented by the Libsvm[5] library [20]. On the large datasets, the QELM is compared to ELM and to the linear kernel support vector machine, henceforth named LSVC, implemented by the Liblinear[6] library [36]. The reason of using LSVC instead of SVC for large datasets is that the latter (i.e., with RBF kernel) is very slow and has two hyper-parameters (see below) whose tuning is not practical in these cases, while LSVC is designed for large datasets and can be executed without no hyper-parameter tuning.

The experiments used 4-fold cross validation, excepting the small dataset `imseg` and the large datasets `shuttle`, `mnist`, `fruits`, `poker`, `physical` and `hepmass`, that already provide two separated train and test sets. On the small datasets, both ELM and SVC perform a grid-search based hyper-parameter tuning, so there is a training set (including 50% of the patterns of each class, randomly selected), a validation and a test set (each one including 25% of the patterns of each class, randomly selected) for each trial. The grid-search proceeds by training the classifier over the four training sets and evaluating its performance over the 4 validation sets, for each combination of hyper-parameter values. The selected combination is the one with the highest average performance over the 4 validation sets. Then, the classifier is trained, using this selected combination, over the four train and validation sets, and the final performance is averaged over the 4 test sets. In the dataset `imseg`, that has separated original train and test sets, the original training set is splitted in a training set, including 2/3 of the patterns of each class, randomly selected, and a validation set, including the remaining 1/3 of the original training patterns of each class. The selected combination of hyper-parameter values is the one that achieves the best performance over the validation set. The final performance is the one achieved by the classifier over the test set after training over the train and validation set with the best combination. The ELM tuned the hyper-parameter H using values from 10 to $\min(N, 500)$ with step 10. Therefore, $H \leq N$ because the ELM performance reduces for $H > N$ (see our previous experiments in section 2.3), and $H \leq 500$ to avoid excessively large values that would slow down ELM. The SVC tuned the hyper-parameters with values in the range $\{2^i\}$, with $i = -5$ to 15 step 2 for the regularization hyper-parameter λ and $i = -15$ to 3 step 2 for the inverse γ of the RBF kernel spread. On the contrary, no hyper-parameter

[5]https://www.csie.ntu.edu.tw/∼cjlin/libsvm (Visited May, 2021)
[6]https://www.csie.ntu.edu.tw/∼cjlin/liblinear (Visited May, 2021)

Table 2.4: Kappa (in %) and time (in seconds) per fold of QELM, ELM and SVC on the small datasets. Each value on the right part of the last row is the average ratio between the times of ELM (or SVC) divided by the times of QELM. A time of 5.23 means 5.23 seconds, while 5:23 means 5 minutes and 23 seconds.

		Kappa (%)			Time (s/fold)		
No.	Dataset	QELM	ELM	SVC	QELM	ELM	SVC
1	tissue	60.2	61.3	55.7	0.003	0.01	0.20
2	hepatitis	26.5	23.8	39.3	0.005	0.02	0.29
3	wine	96.6	96.6	98.3	0.005	0.03	0.36
4	sonar	42.3	48.0	77.8	0.01	0.06	1.44
5	seeds	94.3	94.2	90.7	0.005	0.04	0.29
6	heart	69.7	71.2	66.8	0.007	0.08	0.86
7	imseg	83.5	84.3	90.3	0.08	0.16	0.74
8	ionosphere	60.0	65.5	88.3	0.01	0.15	1.48
9	dermatology	92.1	95.2	94.5	0.03	0.21	5.76
10	voting	90.9	90.9	90.9	0.007	0.26	0.76
11	wdbc	88.2	88.8	90.9	0.03	0.60	2.33
12	synthetic	77.6	90.4	99.0	0.05	0.74	6.04
13	australian	67.7	71.2	72.0	0.03	1.09	3.08
14	pima	42.5	47.0	42.5	0.02	1.35	2.52
15	energy	81.7	93.3	95.2	0.02	1.39	1.51
16	vehicle	69.9	72.1	76.8	0.03	2.16	4.16
17	annealing	94.0	96.1	97.2	0.04	2.58	8.25
18	tic	96.3	96.3	99.8	0.03	3.75	4.80
19	mammograph	62.5	63.5	67.1	0.03	2.10	11.70
20	german	39.2	38.3	39.8	0.04	4.19	15.84
21	msa	86.3	90.1	99.3	0.03	3.56	8.68
22	isolet	85.1	87.4	94.5	1.03	6.75	3:03
23	abalone	30.5	32.0	33.1	0.20	7.31	1:57
24	sat	83.2	84.2	90.6	1.32	10.36	1:44
25	musk	83.9	85.9	99.1	1.84	11.14	4:38
26	usps	88.9	88.1	97.4	3.65	3:54	9:45
27	grid	93.4	94.8	99.4	1.29	12.89	1:33
28	nursery	96.1	92.6	100.0	3.99	16.93	6:51
	Average	74.4	76.5	81.7	–	36.0	153.8

tuning was performed on the large datasets in order to avoid too slow experiments, so there is only one training and one test set per fold. The ELM used $H = 500$, and the LSVC used

$\lambda = 1$. The performance measurement used is the Cohen kappa [19], that takes into account the class unbalancing. The algorithms were programmed in the Octave scientific programing language[7] and executed on a computer with Intel Core i7-9700K CPU (8 cores) at 3.6GHz with 64GB RAM under Kubuntu 20.04.

Table 2.4 reports the kappa and time (left and right parts of the table, respectively) achieved by QELM, ELM and SVC on the small datasets. In the majority of the datasets, SVC achieves the best kappa, overcoming QELM and ELM, with the highest average value (81.7%), 5.2 points above ELM, which is expectable because SVC is an state-of-the-art classifier. A Wilcoxon sign ranksum test comparing the kappa of SVC and ELM reports a p-value of 0.085, so the difference which is not statistically significant. Comparing ELM and QELM, the difference is small (2.1 points) and far from being significant (p-value 0.517), in fact only in two datasets (`synthetic` and `energy`) the difference slightly overcomes 10 points, so QELM follows adequately the performance of ELM. In the small datasets, the number of training patterns N_c of each class is lower than the upper bound L_c on the number of prototypes, so QELM uses all the training patterns as prototypes, and the columns of matrix $\mathbf{Y}$ are the own training patterns, as in ELM. Thus, the only difference between QELM and ELM is that the former does not perform tuning of the hyper-parameter H, so the difference in performance between them measures the impact of this lack of tuning. Since this difference is fairly low (2.1 points in average), the strategy used by QELM to estimate H (eq. 2.1) can be considered quite successful.

Regarding the time required per fold (right part of Table 2.4), QELM is systematically faster than ELM in the small datasets, being the latter between 2 and 142 times slower than the former depending on the dataset. Since the average of times has no statistical sense, because only the larger datasets would contribute to the mean, the right part of the last row reports the average, over all the datasets, of the time of ELM divided by the time of QELM, and the analogous for SVC. Thus, the value in the QELM column is empty (—). This average value reports that ELM is in average 36 times slower than QELM, that performs hyper-parameter tuning by grid-search and therefore repeats the training and test several times. Comparing SVC and QELM, the former is about 154 times slower than the latter, ranging from 9 to 580 times slower depending on the dataset. The three classifiers are slower in dataset `isolet`, compared to the other datasets, due to its high number of classes. In dataset `nursery`, QELM spends 4 s/fold, while ELM spends 17 s/fold and SVC about 7 minutes/fold.

[7]http://www.octave.org (Visited May, 2021)

Table 2.5: Kappa (in %) of QELM, QELM with $H = 500$ (column QELM*), ELM and LSVC, and time (in seconds) per fold of QELM, ELM and LSVC, on the large datasets. Symbol '–' means an execution error. The final average is over datasets where ELM and LSVC executed without errors.

		Kappa (%)				Time (s/fold)		
No.	Dataset	QELM	QELM*	ELM	LSVC	QELM	ELM	LSVC
1	magic	56.6	65.8	67.3	52.5	18.91	1.16	0.24
2	letter	92.7	88.3	88.3	52.9	19.16	1.26	0.60
3	arabic	21.1	23.5	28.0	25.4	1:04	29.91	2:48:09
4	chess	48.0	37.8	39.6	26.9	44.20	2.17	0.69
5	adult	38.0	48.3	50.7	55.6	50.69	9.07	5.31
6	poker	10.6	11.5	16.2	0.1	1:32	30.44	18.67
7	shuttle	98.1	99.0	99.3	46.3	34.00	4.55	0.79
8	wisdm	47.0	42.8	43.5	25.0	1:33	17.25	24.07
9	fruits	32.0	32.0	34.9	–	1:01:51	1:07:14	–
10	devanagari	52.0	39.7	55.6	55.1	2:01	2:20	27:52
11	ijcnn1	62.9	53.9	50.5	22.0	39.35	10.94	3.44
12	covtype	53.1	57.5	59.1	52.7	1:21	1:18	23.84
13	kddcup	87.5	87.5	92.6	92.7	6:33	12:22	3:34
14	record	73.3	–	–	1.8	9:27	–	1:21
15	physical	-0.1	–	–	0.1	11:43	–	7:57
16	baiot	50.0	–	–	-8.2	17:20	–	2:48:05
17	hepmass	22.9	–	–	67.2	15:21	–	5:33
18	higgs	14.7	–	–	27.1	16:08	–	5:41
19	human	17.9	–	–	23.6	16:13	–	23:11
20	kitsune	84.9	–	–	–	25:33	–	–
21	har	0.5	–	–	–	17:40	–	–
22	wesad	2.9	–	–	-2.2	19:31	–	5:25
	Average	55.6	54.6	57.6	42.3	–	0.5	14.4

Table 2.5 reports the kappa and time of QELM, ELM and LSVC on the large datasets. The QELM is able to run in all the datasets, because the size of the matrices **A**, **B** are $H \times I$ and $C \times H$, respectively, while **Y** is of size $H \times M$, with $M \leq N_1$, so none of them scales with the number N of training patterns. However, the ELM fails in 9 of 22 datasets (about 41% of them), while LSVC, which is designed for large-scale datasets, fails in 3 of the largest datasets. Considering performances, QELM also follows ELM in the large datasets, and ELM

overcomes QELM by more than 10 points in 2 datasets (magic and adult), although QELM also overcomes ELM by the same margin in datasets chess and ijcnn1. Comparing average values over the datasets where ELM does not fail, ELM overcomes QELM by only 2 points, and the p-value of the Wilcoxon test is 0.74, so the difference between them is very far from statistical significance.

The difference between ELM and QELM in large datasets is caused by two reasons. 1) Following eq. 2.1, QELM uses $H = 2,250$ neurons on the large datasets, while ELM uses $H = 500$, so the latter is speeded up compared to the former. 2) The QELM uses prototypes while ELM uses training patterns, so the latter saves the time spent by prototype calculation, but at the cost that the former fails in 41% of datasets. In order to evaluate the impact of the second reason (prototyping) over performance, we developed an experiment comparing QELM with $H = 500$ and ELM, that also uses $H = 500$, so prototyping is the only difference between them. The column labeled QELM* in the left part of Table 2.5 reports the kappa of QELM* for the large datasets where ELM did not fail. In the majority of the datasets, excepting devanagari, QELM* performs similarly to ELM and the average kappa of QELM* is 3 points below ELM, near to the average difference between QELM and ELM (2 points). Thus, the use of H=2,250 in QELM with respect to H=500 in QELM* increases performance from 54.6% (QELM*) to 55.6% (QELM).

The performance of LSVC is clearly poorer than QELM and ELM, being 13.3 and 15.3 points below them with p-values (Wilcoxon test) of 0.285 and 0.214, respectively. The results of LSVC are very irregular, e.g. in dataset record LSVC achieves 1.8% while QELM achieves 73.3%. Other cases where LSVC performs poorly are letter, chess, shuttle, wisdm and ijcnn1. However, in hepmass LSVC achieves 67.2% and QELM 22.9%, and in adult LSVC also outperforms QELM. In the largest datasets (numbers 15-22 in Table 2.5), both LSVC and QELM perform poorly, which suggests that they are really difficult datasets.

The times in the right part of Table 2.5 report that QELM is twice slower than ELM. This is caused by: 1) ELM no longer performs hyper-parameter tuning, but uses $H = 500$, what speeds it up; and 2) QELM must calculate prototypes, while ELM uses directly the original training patterns, so it fails in datasets 14-22. On the other hand, the LSVC is about 15 times slower than QELM, because despite of using a fast linear kernel, LSVC still uses the iterative SVC algorithm to solve the classification process, that is slow for large scale datasets. Comparing LSVC and QELM by datasets, in some cases LSVC is faster than QELM, but in

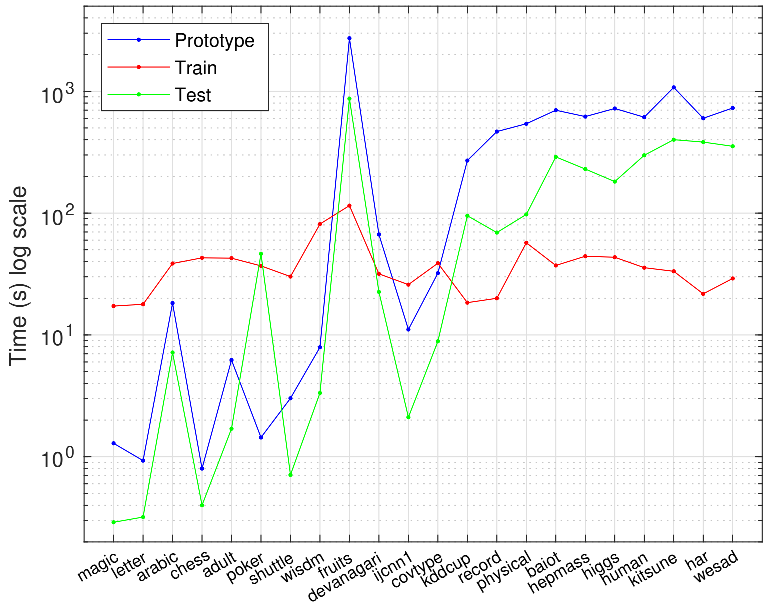


Figure 2.5: Times required by QELM for prototype calculation, train and test in the large datasets sorted by increasing N.

general LSVC is much slower, e.g. in `arabic`, `devanagari` and `baiot`, with high many inputs, failing in dataset `fruits` due to the high $I = 30,000$ and $C = 131$. Note also that times of QELM raise slowly with N from 6 minutes in `kddcup` (4 million patterns) to 19 minutes in `wesad` (31 millions). This is caused by the limitation on the matrix sizes, although increasing on N also raises the time spent by the prototype calculation. There are also some peaks in `kitsune` and `fruits`, due to their high I values (115 and 30,000, respectively).

Figure 2.5 plots the times spent by QELM to calculate prototypes, to perform training (i.e., to calculate the pseudo-inverse of the $\mathbf{Y}$ matrix) and to test, on the large datasets. In the left part, that corresponds to the smaller datasets, the times are lower, and raise when we shift to the right because the dataset size increases, with some peaks in datasets with many inputs (`arabic`, `adult`, `devanagari` and `fruits`). However, on the right part the three times are relatively stable thanks to the bounding on the dimensions $H \leq 2,250$ and $M \leq 5,000$ of the $\mathbf{Y}$ matrix. There is a slow increasing in the prototype time (blue line) because more patterns must be used in the prototype calculation. Once prototypes are created, the train time (red line) also remain stable over all the large datasets. However, the test time (green line) raises because the time required to read the test patterns increases with the size of the test set. Note also the peaks on the test time in datasets `arabic`, `fruits` and `devanagari`, due to the high I, and the peak of `poker`, because this dataset has a reduced training set (N=25,000) but a large test set (P=1,000,000).

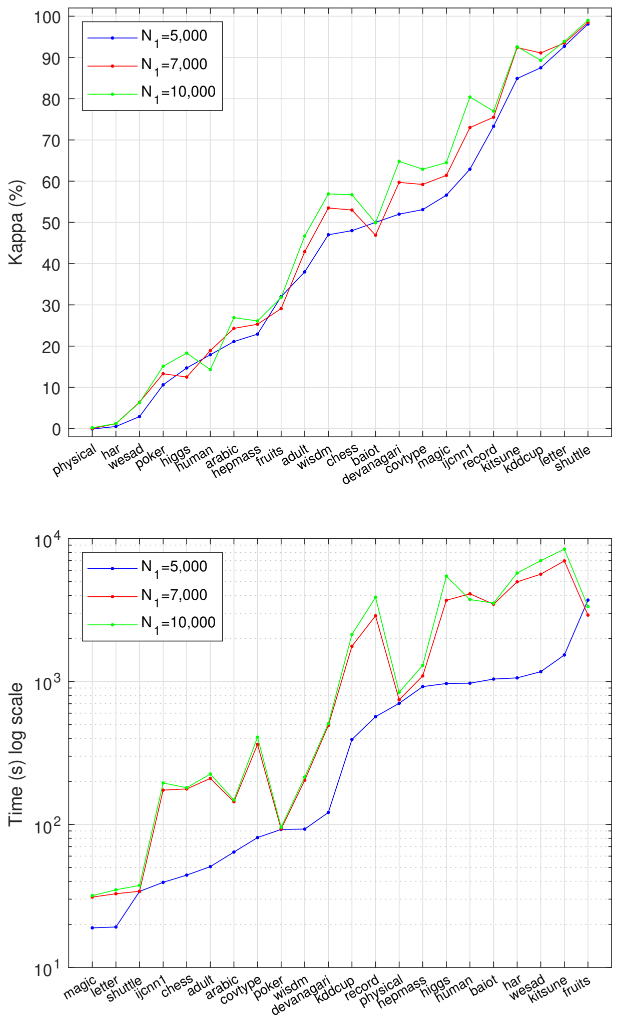


Figure 2.6: Kappa and time, sorted increasingly, achieved by QELM in large datasets varying N_1.

Figure 2.6 plots the kappa and time of QELM on the large datasets with maximum number

Table 2.6: Kappa (in %) of QELM for the large datasets varying the hyper-parameters H, N_0, η and L_0, and average values (asterisk labels the results achieved using the default hyper-parameter values).

	H			N_0			η			L_0		
Dataset	Eq. (1)	5,000	7,000	15,000	20,000	30,000	0.1	0.15	0.2	100	300	500
magic	56.6	24.4	22.7	56.6	56.6	56.6	60.5	56.6	52.1	56.6	56.6	56.6
letter	92.7	21.3	63.2	92.7	92.7	92.7	92.3	92.7	92.4	92.7	93.7	94.1
arabic	21.1	1.0	15.6	21.1	21.1	21.1	20.2	21.1	20.9	21.1	25.7	28.2
chess	48.0	11.5	17.9	48.0	48.0	48.0	47.8	48.0	45.6	48.0	49.4	50.5
adult	38.0	9.9	9.5	38.0	30.5	22.0	42.9	38.0	30.5	38.0	38.0	38.0
poker	10.6	1.3	3.9	10.6	10.1	6.9	11.1	10.6	10.1	10.6	10.6	10.6
shuttle	98.1	97.9	98.1	98.1	98.0	97.7	98.5	98.1	98.0	98.1	98.3	98.6
wisdm	47.0	3.6	4.8	47.0	38.1	9.7	50.1	47.0	38.1	47.0	49.0	56.2
fruits	32.0	25.2	9.2	32.0	30.3	24.0	29.9	32.0	30.3	32.0	23.8	—
devanagari	52.0	1.6	38.7	52.0	48.8	16.2	48.3	52.0	48.8	52.0	68.0	71.2
ijcnn1	62.9	22.4	26.5	62.9	49.6	23.2	79.3	62.9	49.6	62.9	62.9	63.1
covtype	53.1	22.8	22.9	53.1	41.8	22.5	58.9	53.1	41.8	53.1	56.4	57.7
kddcup	87.5	88.2	88.8	87.5	88.2	88.3	89.2	87.5	88.2	87.5	87.5	87.5
record	73.3	75.4	76.4	73.3	72.1	74.4	73.9	73.3	72.1	73.3	39.6	34.3
physical	-0.1	0.0	0.1	-0.1	0.0	-0.1	0.1	-0.1	0.0	-0.1	-0.1	-0.1
baiot	50.0	56.8	58.4	50.0	52.8	56.2	45.5	50.0	52.8	50.0	49.8	46.7
hepmass	22.9	-0.5	12.8	22.9	15.0	7.9	31.2	22.9	15.0	22.9	22.9	22.9
higgs	14.7	0.9	6.9	14.7	12.5	8.1	18.0	14.7	12.5	14.7	14.7	14.7
human	17.9	16.9	16.9	17.9	15.8	15.8	16.8	17.9	15.8	17.9	7.7	11.3
kitsune	84.9	88.1	90.9	84.9	87.6	88.3	85.6	84.9	87.6	84.9	86.0	86.0
har	0.5	1.0	0.5	0.5	-0.2	0.5	2.1	0.5	-0.2	0.5	1.2	1.2
wesad	2.9	4.2	6.3	2.9	2.1	3.3	8.8	2.9	2.1	2.9	2.9	2.9
Average	43.9*	26.1	31.4	43.9*	41.4	35.6	46.0	43.9*	43.9	43.9*	42.9	44.4

of prototypes N_1=5,000 (value used in the previous experiments), N_1=7,000 and N_1=10,000. The kappa is very similar with the different values of N_1, with average values of 43.9%, 46.9% and 48.8% with $N_1 = 5{,}000$, 7,000 and 10,000, respectively. Since increasing N_1 more prototypes are used, the times increase in average 3.13 and 3.57 times for $N_1 = 7{,}000$ and $N_1 = 10{,}000$, respectively, compared to $N_1 = 5{,}000$, with some exceptions. The conclusion is that increasing N_1 above 5,000 only slows QELM down without raising kappa too much, so no tuning of N_1 is required to achieve a good performance.

Table 2.6 reports the kappa achieved by QELM on the large datasets using several values of the hyper-parameters H (number of hidden neurons), N_0 (maximum N for which $H = \lfloor \eta N \rfloor$), η (fraction of N used for H when $N < N_0$) and L_0 (lower limit to the number of prototypes of a class when $C < M_0/L_0$, see subsection 2.4). The missing value for $L_0 = 500$ and dataset `fruits` is because this large value and the high number of classes (131) of dataset `fruits` drive QELM out of memory. The lower row reports the average kappa, and the asterisks identify the column achieved with the default values in Table 2.1 (H given by eq. 2.1, $N_0 = 15{,}000$, $\eta = 0.15$ and $L_0 = 100$). The influence of H is very high (columns 2-4), but eq. 2.1 reports the highest kappa, that confirms the performance reduction for $H > 0.2$, as in Figure 2.4, while QELM is slower because H is higher (not shown in Table 2.6). Regarding N_0 (columns 5-7), performance reduces when its value is increased (see the average kappa), while driving QELM also slower. To change the η value between 0.1 and 0.2 (columns 8-10) has also low influence over performance, although η=0.1 seems to perform slightly better. Similarly, to increase L_0 (columns 11-13) slows down QELM, because more prototypes must be calculated, but does not raise appreciably performance with respect to $L_0 = 100$ (default value). Overall, these results show that QELM is not very sensitive to the choice of N_1 (see Figure 2.6), N_0, η and L_0, so their values do not need to be tuned. The QELM is sensitive to H, but raising its value compared to the value set by eq. 2.1 only reduces performance.

CHAPTER 3

EXTREME LEARNING MACHINE WITH CONFIDENCE INTERVAL BASED BIAS INITIALIZATION

The randomness of input weights and bias in the ELM algorithm attracted much attention because its simplicity, accelerated learning and generalization performance compared to previous iterative training methods. However, despite ELM uses random values for the input weights and a closed-form expression to calculate the output weights, some researchers have proposed more complex variants that pursue to select optimal weight values and number of hidden neurons. For example, Son [107] proposed an improved training method for ELM with genetic algorithms using selection, crossover and mutation, which randomly selects hidden layer weights and biases. Zhou [130] proposed a hybrid grey wolf optimization algorithm based on fuzzy weights and differential evolution. The projection vector machine [29] uses singular value decomposition (SVD) to calculate the ELM weights, differently from the random weights in ELM, and the rank of the SVD matrix is the number of hidden neurons. A fast training algorithm is proposed in [1], that uses efficient prediction sum of squares criteria and SVD to calculate the weights of a regularized ELM ensemble.

There are also works in the literature where ELM uses pruning to select the best hidden neurons [40, 96]. The bootstrapped least absolute shrinkage and selection operator (LASSO) ELM [27] also uses pruning, regularization and resampling to select the most representative hidden neurons. Growing algorithms instead of pruning were also proposed [37], combined

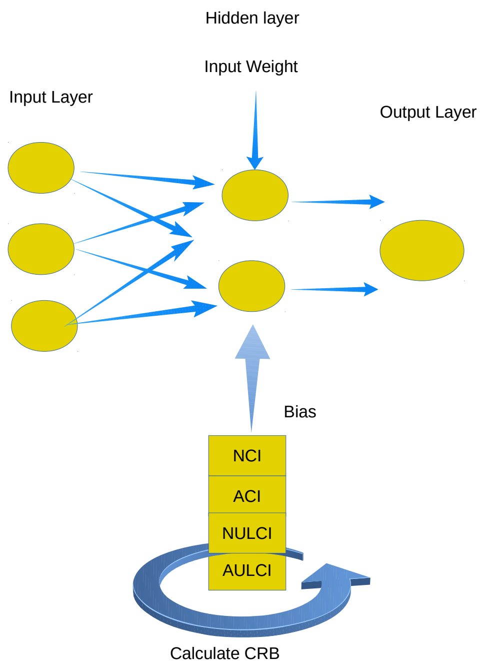


Figure 3.1: Structure of the proposed method CRB-ELM.

with incremental learning, for the ELM training. Huang [53] also proposed an incremental ELM that uses random search to add new hidden neurons. To find the parameters in the layers, Qu [94] proposed a two-hidden-layer ELM to obtaining the parameters between the first and second hidden layer. The work [121] introduces an algorithm that determines randomly the input weights and biases based on the Liu regression estimator. This approach handles drawbacks like instability and poor generalization performance in the presence of perturbations and multicollinearity. The collaborative ELM [97] enhances the performance and speed by considering where plausible predictions are wrong, within the desired degree of confidence, thus avoiding unnecessary calculations in the network neurons. This model was applied to diagnose of clinical events by analyzing patients' medical records, achieving high accuracy (up to 98%). The confidence interval for the ELM weights is proposed in [101] to present prediction points, and keeps a multivariate normal distribution over the output weight vector. To achieve the minimum informative distribution and maintain the forecast confidence interval, Akusok [7] proposed to estimate the input-dependent prediction intervals by a separate ELM model, and in [6] proposed to use the confidence interval in order to make predictions for each data sample and to make more interpretable the ELM results. Lai [65] proposed a regularized ELM using the biased drop connect and biased dropout to set the weights so that generalization performance is raised and overfitting is reduced.

This chapter proposes the confidence random bias extreme learning machine (CRB-ELM)[1], that enhances the classical ELM by defining a confidence level used to estimate a confidence interval. Both confidence interval and level are used to calculate the random bias of the hidden neurons without computationally intensive computations. The results demonstrate that calculating the bias in the proposed way, taking into account the training data, enhances performance both in classification and regression task [13]. Section 3.1 introduces the confidence interval and level, while sections 3.2 and 3.3 describe the proposed models CRB-ELM and CRB-SVD-ELM. The experimental work is discussed in sections 3.4 and 3.5.

3.1 Confidence interval and confidence level

The confidence interval (CI) is a statistics measurement that estimates the interval where the sample input data ($\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^{N} = \{x_{ni}\}_{ni=1}^{N,I}$) are located [106], being an alternative to standard deviation of the data (denoted as σ). In this work, we computed CI for the bias $\{d_h\}_{h=1}^{H}$ of the hidden neurons from the input data $\mathbf{X}$. The lower and upper limits L_d and U_d, respectively, of the confidence interval $[L_d, U_d]$ for bias d_h are defined as:

$$L_d = \mu - \frac{\sigma Z_\alpha}{\sqrt{2}}, \quad U_d = \mu + \frac{\sigma Z_\alpha}{\sqrt{2}} \tag{3.1}$$

where μ and σ are the mean and standard deviation of the input data $\mathbf{X}$:

$$\mu = \frac{1}{NI}\sum_{n=1}^{N}\sum_{i=1}^{I} x_{ni}, \quad \sigma = \sqrt{\frac{1}{NI}\sum_{n=1}^{N}\sum_{i=1}^{I}(x_{ni}-\mu)^2} \tag{3.2}$$

while Z_α is the z-score:

[1]Code link: https://github.com/owdaybtoush

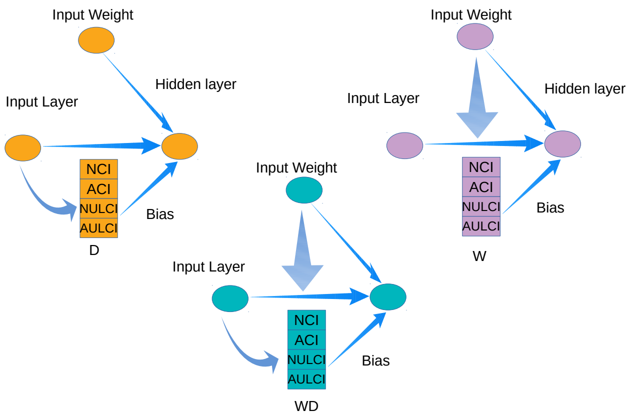


Figure 3.2: The three input data (D,W and WD).

$$Z_\alpha = \Phi^{-1}\left(1-\frac{\alpha}{2}\right), \quad 0<\alpha<1 \tag{3.3}$$

being $\Phi(x)$ the cumulative distribution function of the standard normal distribution:

$$\Phi(x) = \frac{1}{\sigma\sqrt{2}}\int_{-\infty}^{x} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right)dt \tag{3.4}$$

In our approach, named confidence random bias extreme learning machine (CRB-ELM), the biases $\{d_h\}_{h=1}^{H}$ are randomly calculated using four different methods to split the data population (see Fig. 3.1 for a schematic diagram of CRB-ELM and these four methods). These methods use:

1. Original input data (named as normal confidence interval, NCI).
2. Absolute value input data (absolute confidence interval, ACI).
3. Upper-lower bound of the CI (normal upper-lower confidence interval, NULCI), with intervals $[L,0]$ and $[U,1]$.
4. Absolute upper-lower bound of the confidence interval (AULCI), with intervals $[L_d,0]$ and $[U_d,1]$.

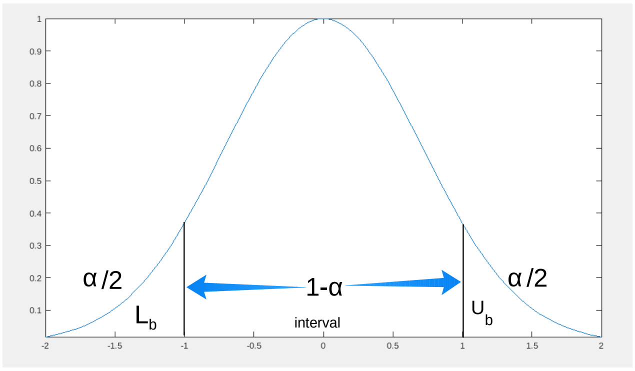


Figure 3.3: Graphical meaning of the confidence level.

Our experiments tested the optimal CRB-ELM with three different input data (see Fig. 3.2):

1. Data only (named D), i.e., matrix $\mathbf{X}$.
2. Input weights only (W), i.e., matrix $\mathbf{A}$.
3. Multiplying the input weights by the data (named WD), i.e., $\mathbf{AX}^T$.

The confidence level, denoted as F, is defined as the probability (in %) that the data are located between the lower and upper intervals (in our case L_d and U_d, respectively) of the confidence interval. Thus, the probability of data are located outside this interval is 100α, assuming a large sample size N. Mathematically, F can be calculated as $F = 100(1-\alpha)$. Fig 3.3 shows the meaning of F. The most commonly used F and z-score values are shown in Table 3.1. As an example, for F=95.9% the table sets Z_α=1.960. The lower and upper bounds L_d and U_d can be calculated using eq. 3.1. Using these values, only 4.1% of data are allowed to fall outside the confidence interval $[L_d, U_d]$.

Table 3.1: Critical values commonly used for the confidence level F for μ=0 and σ=1.

F	Z_α
90.9	1.645
95.9	1.960
97.9	2.750
99.9	3.291

3.2 Confidence random bias extreme learning machine

The proposed method CRB-ELM proceeds as follows. First, the confidence level F is set from Table 3.1, and the lower and upper bounds L_d and U_d of the confidence interval are calculated according eq. 3.1. Then, the number H of hidden neurons and the activation function $g(x)$ are selected (eqs. 1.7-1.11 in chapter 1), and the input weights $\{a_{hi}\}_{hi=1}^{H,I}$ and bias $\{d_h\}_{h=1}^{H}$ are randomly generated within the confidence interval $[L_d, U_d]$. Specifically, we use values for H from the set $\{$3, 5 ,7, 10, 12 ,15 ,25, 40, 50, 65, 70, 80, 90, 100 ,110, 125, 140, 150, 170, 200$\}$, values for F from Table 3.1, and activation function from eqs. 1.7-1.11. Once the values for H, F and g are selected, the test stage of ELM is executed and the performance is evaluated. Algorithm 4 compiles the whole CRB-ELM method.

3.3 Confidence random bias-singular value decomposition ELM

Singular value decomposition (SVD) is a factorization technique that decomposes a rectangular matrix as the product of three matrices. This method is useful for reducing the data dimensionality, and it has been applied in artificial neural networks and machine learning. The goal of SVD is to decompose a $(m \times n)$-order matrix $\mathbf{A}$ as $\mathbf{USV}^T$. Matrices $\mathbf{U}$ and $\mathbf{V}$ are squared and orthogonal containing the right and left singular vectors of the original $\mathbf{A}$ matrix, while $\mathbf{S}$ is a diagonal matrix with the singular values of $\mathbf{A}$.

In the current research, we propose the confidence random bias-singular value decomposition extreme learning machine (CRB-SVD-ELM), that uses the SVD method to decompose the matrix $\mathbf{X} = [\mathbf{1}_N \; \{x_{ni}\}_{ni=1}^{N,I}]$, where $\mathbf{1}_N$ is the column vector of length N with ones. This augmented data matrix $\mathbf{X}$ is of order $N \times (I+1)$. Thus:

$$\mathbf{X} = \mathbf{USV}^T \tag{3.5}$$

Algorithm 4: Confidence random bias extreme learning machine.

```
1 Algorithm: [A, B]=CRB-ELM(X,T,H,g,F)
  Data: $\mathbf{X} = \{x_{ni}\}_{ni=1}^{N,I}$: training set; $\mathbf{T} = \{t_{cn}\}_{cn=1}^{C,N}$: true output; $H$: number of hidden
      neurons; $g$: activation function; $F$: confidence level
  Result: $\mathbf{A} = \{a_{hi}\}_{hi=1}^{H,I}$: input weights; $\mathbf{d} = \{d_h\}_{h=1}^{H}$: bias; $\mathbf{B} = \{b_{ch}\}_{ch=1}^{C,H}$: output
      weights.
2 $\mu \leftarrow \frac{1}{NI}\sum_{n=1}^{N}\sum_{i=1}^{I} x_{ni}$; $\sigma \leftarrow \sqrt{\frac{1}{NI}\sum_{n=1}^{N}\sum_{i=1}^{I}(x_{ni}-\mu)^2}$.
3 $\Phi(x) \leftarrow \frac{1}{\sigma\sqrt{2}}\int_{-\infty}^{x} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right) dt$.
4 $\alpha \leftarrow 1 - \frac{F}{100}$; $Z_\alpha \leftarrow \Phi^{-1}\left(1-\frac{\alpha}{2}\right)$.
5 $L_d \leftarrow \mu - \frac{\sigma Z_\alpha}{\sqrt{2}}$; $U_d \leftarrow \mu + \frac{\sigma Z_\alpha}{\sqrt{2}}$.
6 Input weight $\mathbf{A}$ initialized with random values.
7 Bias $\mathbf{d}$ initialized with random values in $[L_d, U_d]$.
8 Hidden layer activity: $\mathbf{Y} = \{y_{hn}\}_{hn=1}^{H,N} \leftarrow \{g(\mathbf{a}_h^T \mathbf{x}_n + d_h)\}_{hn=1}^{H,N}$.
9 Output weight: $\mathbf{B} \leftarrow \mathbf{T}\mathbf{Y}^\dagger$.
```

The CRB-SVD-ELM proceeds by decomposing **X** and using the **V** matrix to calculate the confidence interval for the random generation of the hidden layer bias $\{d_h\}_{h=1}^{H}$. Similarly to CRB-ELM, the data matrix **X** can alternatively be replaced by the input weight **A** or by the weight-data product $\mathbf{AX}^T$ (variants W and WD in section 3.1). From this point, CRB-SVD-ELM follows the same steps as CRB-ELM, as reported in algorithm 5.

The proposed methods CRB-ELM and CRB-SVD-ELM are compared in sections 3.4 and 3.5 to the classical ELM. To enrich this comparison, we will also include the base projection vector machine (BPVM), that uses SVD as input weights, while bias are random values [29] and the number H of hidden neurons is set to the rank of matrix **S**. Fig. 3.4 describes the differences between BPVM and the proposed method CRB-SVD-ELM, summarized by algorithm 5. In order to make its comparison to BPVM fair, the CRB-SVD-ELM also sets H to the rank of the matrix **S**. We used the BPVM implementation [29] and the classical ELM code [57] publicly available. The BPVM used sigmoid activation function $g(x) = 1/(1+\mathrm{e}^{-\alpha x})$ with α=[1 0.5 0.05 0.01]. In the experiments, all the inputs have been normalized into the range [-1, 1].

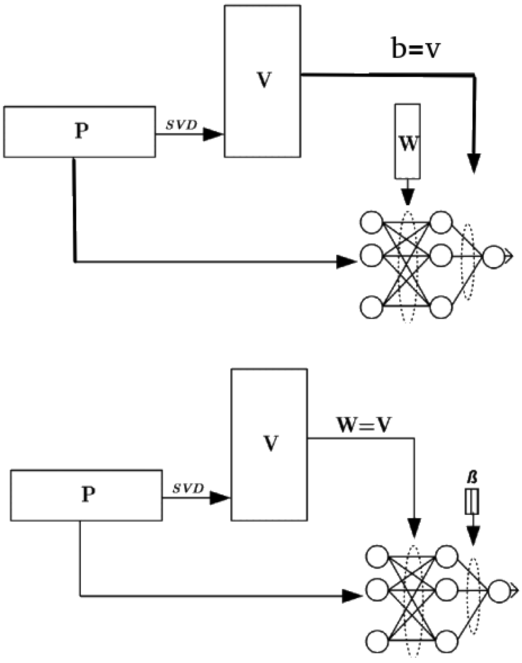


Figure 3.4: Block diagrams of CRB-ELM (upper panel) and BPVM (lower panel (29)).

3.4 Classification results

The experimental work compares CRB-ELM and CRB-SVD-ELM with the classical ELM and BPVM over a collection of 27 classification datasets selected from the UCI Machine Learning Repository[2], whose specifications (numbers N and I of patterns and inputs, respectively) are listed in Table 3.2. The CRB-ELM and CRB-SVD-ELM were codified under MatLab R2018. All the experiment were executed under a computer with 8 Intel Core i7-4790k processors at 4GHz, with 16 GB RAM and operative system Ubuntu 18.04. We splitted the dataset into training, validation and test sets, and the performance was evaluated using the standard 4-fold cross-validation in our experiments. The CRB-ELM used the four splitting methods (NCI, ACI, NULCI, AULCI) and the three input types (WD, W and D) listed in

[2]https://archive.ics.uci.edu (Visited May, 2021)

Algorithm 5: Confidence random bias-singular value decomposition ELM.

```
1  Algorithm: [A, B]=CRB-SVD-ELM(X,T,H,g,F)
   Data: $\mathbf{X} = [\mathbf{1}_N\ \{x_{ni}\}_{ni=1}^{N,I}]$: training set; $\mathbf{T} = \{t_{cn}\}_{cn=1}^{C,N}$: true output; $H$: number of
         hidden neurons; $g$: activation function
   Result: $\mathbf{A} = \{a_{hi}\}_{hi=1}^{H,I}$: input weights; $\mathbf{d} = \{d_h\}_{h=1}^{H}$; $\mathbf{B} = \{b_{ch}\}_{ch=1}^{C,H}$: output weights.
2  $\mathbf{X} = \mathbf{USV}^T$. Let be $\mathbf{V} = \{v_{ni}\}_{ni=1}^{N,p}$ //Perform SVD of $\mathbf{X}$ as $\mathbf{USV}^T$
3  // Use $\mathbf{V}$ to calculate the CI for the random generation of bias $\mathbf{d}$.
4  $\mu \leftarrow \frac{1}{N(I+1)} \sum_{n=1}^{N} \sum_{i=1}^{I+1} v_{ni}$; $\sigma \leftarrow \sqrt{\frac{1}{N(I+1)} \sum_{n=1}^{N} \sum_{i=1}^{I+1} (v_{ni} - \mu)^2}$.
5  $\Phi(x) \leftarrow \frac{1}{\sigma\sqrt{2}} \int_{-\infty}^{x} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right) dt$.
6  $\alpha \leftarrow 1 - \frac{F}{100}$; $Z_\alpha \leftarrow \Phi^{-1}\left(1 - \frac{\alpha}{2}\right)$.
7  $L_d \leftarrow \mu - \frac{\sigma Z_\alpha}{\sqrt{2}}$; $U_d \leftarrow \mu + \frac{\sigma Z_\alpha}{\sqrt{2}}$.
8  Bias $\mathbf{d}$ initialized with random values in $[L_d, U_d]$.
9  Hidden layer activity: $\mathbf{Y} = \{y_{hn}\}_{hn=1}^{H,N} \leftarrow \{g(\mathbf{a}_h^T \mathbf{v}_n + d_h)\}_{hn=1}^{H,N}$.
10 Output weight: $\mathbf{B} \leftarrow \mathbf{TY}^\dagger$.
```

section 3.1. The performance measurement used was accuracy (ACC) in %, defined as:

$$ACC(\%) = \frac{100}{N} \sum_{n=1}^{N} \delta(t_n, z_n) \tag{3.6}$$
$$t_n = \arg\max_{c=1,\ldots,C}\{t_{cn}\}, \quad z_n = \arg\max_{c=1,\ldots,C}\{v_{cn}\}$$

where t_n and z_n are the true and predicted output for the n-th test pattern, calculated respectively as the argument that maximizes $\{t_{cn}\}$ and $\{v_{cn}\}$, respectively, for $c = 1, \ldots, C$ (see the notation in table 2.2 and the figure 2.1 in chapter 2). Besides, $\delta(t_n, z_n) = 1$ when $t_n = z_n$ and $\delta(t_n, z_n) = 0$ otherwise. In the experiments, the values of the confidence level F in Table 3.1 were used, and the value with the highest average ACC over the training set was selected.

Table 3.3 reports the accuracy of CRB-ELM and ELM on the classification datasets with the three input types WD, W and D (in bold the best accuracy for each dataset). The results indicate the superiority of the three methods WD, W and D over ELM in average with similar deviation. Comparing the three variants of CRB-ELM, the version that uses input

Table 3.2: Collection of UCI classification datasets (sorted albabetically by dataset name): number of patterns N, inputs (I) and classes (C).

Original Name	Dataset	N	I	C
Abalone	abalone	4,177	8	3
Australian Sign Language signs	australian	6,650	15	2
Chess(King-Rook vs. King)	chess	18,056	6	18
Connect-4	connect-4	67,557	42	3
Energy efficiency	energy-heat	768	8	3
South German Credit	german	1,000	21	2
SPECT Heart	heart	267	22	2
Hepatitis C Virus (HCV)	hcv	1,385	29	2
Image Segmentation	imseg	2,310	19	7
Ionosphere	ionosphere	351	34	2
Letter Recognition	letter	20,000	16	26
MAGIC Gamma Telescope	magic	19,020	11	2
MammographicMass	mammograph	961	6	2
Miniboone particle identification	miniboone	130,065	49	2
MONK's Problems	monks2	432	7	2
MicroMass	msa	931	1300	20
Nursery	nursery	12,960	8	4
Pima Indians Diabetes data	pima	768	8	2
Planning Relax	planning	182	13	2
Seeds	seeds	210	7	3
Shuttle Landing Control	shuttle	57,977	6	7
Connectionist Bench	sonar	208	60	2
Synthetic Control Chart	synthetic	600	60	6
Tic-Tac-Toe Endgame	tictac	958	9	2
Statlog (Vehicle Silhouettes)	vehicle	946	18	4
Congressional Voting Records	voting	435	16	2
Wine	wine	178	13	3

data to estimate bias (D) is clearly better, achieving the best accuracy on all the datasets, while the poorest results are achieved by W (only weights), although differences among the three methods are not very large. The ELM achieves the best accuracy in 5 of 27 datasets (`ionosphere`, `letters`, `shuttle`, `magic` and `sonar`), but in all of them some variant of CRB-ELM also achieves the same accuracy.

The BPVM uses SVD to calculate the weights and bias, instead of the random initialization of ELM, while CRB-SVD-ELM uses the **V** matrix provided by SVD as input data to

Table 3.3: Accuracy of ELM and CRB-ELM using input weights multiplied by data (WD), weights (W) and data (D).

		CRB-ELM		
Dataset	ELM	WD	W	D
abalone	65.33	66.10	**66.32**	**66.32**
australian	85.94	**86.37**	86.23	**86.37**
chess	34.91	**35.08**	**35.01**	**35.08**
connect-4	72.60	72.89	**72.93**	**72.93**
energy-heat	92.19	92.97	**93.09**	**93.09**
german	66.39	68.59	**69.67**	**69.67**
heart	71.19	71.89	**72.27**	**72.27**
hepatitis	78.68	**84.46**	78.09	**84.46**
imseg	92.48	**93.05**	92.76	**93.05**
ionosphere	**83.72**	82.56	81.44	**83.72**
letter	**72.05**	71.97	71.78	**72.05**
magic	**83.20**	83.19	83.12	**83.20**
mammograph	76.60	**76.92**	**76.93**	**76.93**
miniboone	89.55	**89.86**	**89.66**	**89.86**
monks2	52.96	63.85	**68.53**	**68.53**
msa	99.67	99.00	**100.00**	**100.00**
nursery	90.08	**91.37**	90.15	**91.37**
pima	73.83	73.70	**75.78**	**75.78**
planning	53.86	**59.63**	53.81	**59.63**
seeds	93.87	94.39	**95.25**	**95.25**
shuttle	**96.50**	96.35	96.35	**96.50**
sonar	**78.98**	78.94	78.47	**78.98**
synthetic	94.83	96.17	**96.33**	**96.33**
tictac	97.08	97.91	**98.23**	**98.23**
vehicle	79.78	79.40	**80.34**	**80.34**
voting	90.88	93.02	**93.26**	**93.26**
wine	96.09	**98.31**	**98.31**	**98.31**
Avg.	80.12	81.41	81.26	**81.91**
Std.	15.68	14.88	15.06	**14.71**

Table 3.4: Comparison of BPVM and CRB-SVD-ELM for classification.

Dataset	BPVM	CRB-SVD-ELM
abalone	60.09	**60.95**
australian	85.22	**85.36**
energy-heat	**82.81**	81.38
german	72.14	**73.5**
heart	**82.6**	81.87
hepatitis	**87.1**	85.78
imseg	81.65	**82.69**
letter	51.26	**51.52**
magic	74.32	**75.65**
monks2	61.63	**62.77**
msa	33.33	**34.35**
nursery	**88.46**	87.38
pima	**72.79**	71.48
planning	70.51	**71.07**
seeds	94.68	**96.18**
sonar	**75.71**	74.14
synthetic	84.83	**90.83**
tictac	**95.92**	92.59
vehicle	71.62	**71.73**
voting	93.88	**93.92**
wine	**97.75**	**97.75**
Avg.	77.06	**77.28**
Std.	15.89	**15.64**

calculate confidence interval for the bias and the weight random initialization. On the other hand, CRB-SVD-ELM uses the rank of matrix $\mathbf{S}$, also provided by SVD, as value for the number H of hidden neurons in ELM. Table 3.4 reports the performances of both methods: CRB-ELM overcomes BPVM in 14 of 21 databases, which represents 67% of the datasets. In the 5 largest datasets (`shuttle`, `connect-4`, `letter`, `chess` and `magic`) the SVD runs into out-of-memory errors. The CRB-ELM also outperforms BPVM in average accuracy with lower deviation.

Comparing the ELM and the best CRB-ELM results (considering the variants WD, W, D, NCI, ACI, NULCI and AULCI, described in section 3.1), Table 3.5 reports an improved performance of CRB-ELM over ELM, that is outperformed in 24 of 27 classification datasets, achieving similar performance in the remaining datasets. These tests demonstrate the robust-

Table 3.5: Accuracy of ELM and the best CRB-ELM.

Number	Dataset	ELM	CRB-ELM
1	abalone	65.33	**66.32**
2	australian	85.94	**86.37**
3	energy-heat	92.19	**93.10**
4	german	66.39	**69.67**
5	heart	71.19	**72.27**
6	hepatitis	78.68	**84.46**
7	imseg	92.48	**93.05**
8	ionosphere	**83.72**	82.56
9	letter	72.05	**72.18**
10	magic	**83.20**	83.19
11	monks2	52.96	**73.77**
12	msa	99.67	**100.0**
13	nursery	90.08	**91.37**
14	pima	73.83	**75.78**
15	planning	53.86	**59.63**
16	seeds	93.87	**95.72**
17	sonar	78.98	**78.94**
18	synthetic	94.83	**96.33**
19	tictac	97.08	**98.23**
20	vehicle	79.78	**80.34**
21	voting	90.88	**93.26**
22	wine	96.09	**98.31**
23	chess	34.91	**35.26**
24	connect-4	72.60	**72.93**
25	mammograph	76.60	**76.93**
26	miniboone	89.55	**89.86**
27	shuttle	**96.50**	96.35
	Avg.	80.12	**82.08**
	Std.	15.68	**14.55**

ness of CRB-ELM performance and the overall improvement with respect to ELM.

3.5 Regression results

The CRB-ELM and CRB-SVD-ELM were also tested in the 18 regression datasets listed in Table 3.6, also selected from the UCI Machine Learning Repository. In this case, the performance measurement was the root mean squared error (RMSE), defined as:

Table 3.6: Collection of UCI regression datasets (sorted alphabetically by dataset name): number of patterns (N) and inputs (I).

Original Name	Dataset	N	I
Airfoil Self-Noise	airfoil	1,503	6
Beijing Multi-Site Air-Quality	air-quality-NOx	420,768	18
Appliances energy prediction	app-energy	19,735	29
Auto MPG	auto-mpg	398	8
Bike Sharing Dataset	bike-day	17,389	16
Combined Cycle Power Plant	combined-cycle	9,568	4
Communities and Crime	com-crime	1,994	128
Concrete Compressive Strength	compress-stren	1,030	9
Daily Demand Forecasting	daily-demand	60	13
Energy efficiency	energy-cool	768	8
Facebook metrics	fb-metrics	500	19
Geo-Magnetic field and WLAN	geo-long	153,540	25
Greenhouse Gas Observing Net	housing	2,921	5,232
Servo	servo	167	4
Concrete Slump Test	slump	103	10
Istanbul stock exchange	stock-exchange	536	8
Student Performance	student-por	649	33
Yacht Hydrodynamics	yacht-hydro	308	7

$$RMSE = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(t_n - z_n)^2} \tag{3.7}$$

where t_n and z_n were already defined in eq. 3.6 of section 3.4. Similarly to classification, we tried the F values listed in Table 3.1, selecting for testing the value with the lowest average RMSE over the training set. In order to compare the proposed method CRB-ELM and ELM, Table 3.7 reports the RMSE achieved by CRB-ELM with the three input methods and ELM (WD, W and D). The CRB-ELM achieves lower error than ELM excepting 3 of 18 databases, `app-energy`, `com-crime` and `student-por`, where the four methods achieve the same error. Considering the average RMSE over all the regression datasets, CRB-ELM achieves the lowest value (0.60) using weights as input data (W), much below ELM (0.74), with even lower deviation (0.36 vs. 0.46 with ELM). However, the three variants of CRB-ELM clearly outperform ELM in terms of average RMSE (0.63, 0.6 and 0.65 vs. 0.74).

Comparing CRB-SVD-ELM to BPVM on the regression datasets, the RMSE values in Ta-

Table 3.7: Values of RMSE achieved by ELM and CRB-ELM using WD, W and D on the regression datasets.

		CRB-ELM		
Dataset	ELM	WD	W	D
airfoil	0.53	0.53	0.53	**0.52**
air-quality-NOx	1.18	**0.41**	0.44	1.01
app-energy	**0.91**	**0.91**	**0.91**	**0.91**
auto-mpg	0.61	**0.57**	0.64	0.68
bike-day	0.20	**0.18**	0.20	0.19
combined-cycle	0.26	**0.25**	**0.25**	**0.25**
com-crime	**0.61**	**0.61**	**0.61**	**0.61**
compress-stren	0.48	0.47	0.47	**0.46**
daily-demand	1.02	0.33	**0.31**	0.48
energy-cool	0.24	**0.24**	0.25	**0.24**
fb-metrics	0.39	0.38	**0.32**	0.38
geo-long	0.93	**0.91**	**0.91**	**0.91**
housing	0.54	**0.46**	0.55	0.53
servo	1.68	2.00	**1.40**	**1.40**
slump	1.72	**1.07**	1.15	1.22
stock-exchange	0.84	0.76	0.72	**0.70**
student-por	**0.99**	**0.99**	1.00	**0.99**
yacht-hydro	0.20	0.18	**0.10**	0.14
Avg.	0.74	0.63	**0.60**	0.65
Std.	0.46	0.44	**0.36**	**0.36**

ble 3.8 shows that CRB-SVD-ELM overcomes BPVM in 7 of 17 datasets, with similar RMSE in the other datasets. The superiority is most evident in datasets `air-quality-NOx`, where the RMSE of BPVM and CRB-SVD-ELM were 0.73 and 0.64, respectively, and `servo`, with 0.65 and 0.60, respectively. The average RMSE values over all the regression datasets were 0.64 and 0.62 for BPVM and CRB-SVD-ELM, respectively, with the same deviation (0.21). The methods BPVM and CRB-SVD-ELM failed on the `yatch-hydro` dataset due to an error on the singular value decomposition, so their results are missing in Table 3.8 for this dataset.

Table 3.9 reports the RMSE achieved by ELM and by the best CRB-ELM (D, W and WD, and NCI, ACI, NULCI and AULCI) over all the regression datasets. The RMSE is lower for the CRB-ELM compared to ELM, that is outperformed in 14 of 18 regression problems, achieving similar performance in the remaining datasets. The average RMSE is also lower for CRB-ELM.

Table 3.8: Values of RMSE achieved by BPVM and CRB-SVD-ELM on the regression datasets.

Dataset	BPVM	CRB-SVD-ELM
airfoil	**0.87**	**0.87**
air-quality-NOx	0.73	**0.64**
app-energy	**0.95**	**0.95**
auto-mpg	**0.46**	**0.46**
bike-day	**0.31**	**0.31**
combined-cycle	**0.47**	**0.47**
com-crime	**0.61**	**0.61**
compress-stren	0.79	**0.75**
daily-demand	**0.42**	**0.42**
energy-cool	**0.35**	**0.35**
fb-metrics	0.38	**0.37**
geo-long	**0.93**	**0.93**
housing	0.67	**0.66**
servo	0.65	**0.60**
slump	**0.88**	**0.88**
stock-exchange	0.71	**0.70**
student-por	0.64	**0.62**
Avg.	0.64	**0.62**
Std.	0.21	**0.21**

Table 3.9: Values of RMSE achieved by ELM and the best CRB-ELM.

Number	Dataset	ELM	CRB-ELM
1	airfoil	0.53	**0.52**
2	air-quality-NOx	1.18	**0.41**
3	app-energy	**0.91**	**0.91**
4	auto-mpg	0.61	**0.57**
5	bike-day	0.20	**0.18**
6	combined-cycle	0.26	**0.25**
7	com-crime	**0.61**	**0.61**
8	compress-stren	0.48	**0.46**
9	daily-demand	1.02	**0.31**
10	energy-cool	**0.24**	**0.24**
11	fb-metrics	0.39	**0.32**
12	geo-long	0.93	**0.91**
13	housing	0.54	**0.46**
14	servo	1.68	**1.40**
15	slump	1.72	**1.07**
16	stock-exchange	0.84	**0.70**
17	student-por	**0.99**	**0.99**
18	yacht-hydro	0.20	**0.10**
	Avg.	0.74	**0.58**
	Std.	0.46	**0.35**

CHAPTER 4

QUICK HIDDEN LAYER SIZE TUNING IN EXTREME LEARNING MACHINE FOR CLASSIFICATION

Chapter 2 of the current memory already stated that one of the limitations of the extreme learning machine is the difficulty to set an appropiated value for the number H of hidden neurons. Very high or low H drives the network into overfitting or underfitting problems [18], and in the first case the time spent by training may also be too high [65]. In order to set the number of hidden neurons and the values of the input weights for the ELM, there are basically three main groups of strategies [134]:

- **Random generation**. This the usual approach, where the hidden neurons are randomly generated, at the same time and independently of the training data [52, 54, 57]. Alternatively, a group of new random neurons is created each time [53], adding to the network only the most informative neurons of the group.

- **Constructive**. The random hidden neurons are added incrementally, one by one, to the network. After each adition, the network is trained again and the best neurons are choosen [54]. The training stops when a maximum number of hidden neurons is reached [52, 53]. This method is unsuitable for real applications because it is time-consuming [66, 134] and does not lead to the optimal network structure [37, 67].

- **Pruning**. The network starts with a large number of hidden neurons, removing iteratively the ones that are less informative according to some statistical criteria, such as sparse regression [105], multiple response [88] or relevance to the class label using chi-squared test and information gain [96].

Alternative approaches include: to update the weights towards the direction that reduces the squared error [122], although these methods are very similar to gradient descent techniques, that are far from the ELM original spirit; to use genetic algorithms for selecting H, input weights and biases; to use particle swarm optimization [59] or singular value decomposition (SVD) parsing [28] to select an appropiate H. Finally, the value of H can also be fine-tuned as a hyper-parameter using grid-search or similar methods.

The previous works attempt to improve ELM generation but encountered some shortcomings such as time-consuming, overfitting, underfitting, or no specific approach to the objective function. Besides, they focused too much on iterative or incremental attempts to improve hidden the neuron weights, leading to approaches with high computational complexity that are undesirable in real applications. The current chapter proposes methods that use statistical criteria such as moving and exponential moving average (MA and EMA), and divide-and-conquer (DC) strategy, to choose H that optimizes the network performance. The experimental work compares the proposed methods, named MA-ELM, EMA-ELM and CD-ELM, to the classical ELM and to the constrained ELM [133], that also provide a solution to make more efficient the tuning of H. Sections 4.1-4.3 describe the proposed methods, while the experimental results are reported and discussed in section 4.4.

4.1 Moving average extreme learning machine

The relationship between the number H of hidden layer neurons and the network performance is one of the more complex problems in ELM, and it is not subject to a specific statistical distribution or type of non-linearity, so we propose to use moving average with ELM in order to perform the tuning of H in the training phase. Moving average (MA) is a simple statistical tool and indicator for technical analysis that makes extensive use of the points to be estimated. It has wide applications in science, engineering, financial applications, time series analysis [83], and neural networks [86]. The MA creates an averaging of a set of user-defined and constantly updated values, in our case the training accuracy of ELM using different values of H. The objective is to smooth out trends in values by filtering out noise from fluctuations of extreme values. The n-width moving average of a value A_k, for $k = 1, 2, \ldots$, denoted as $\mathscr{M}_{k+1}$, is defined by eq. 4.1:

$$\mathscr{M}_{k+1} = \frac{A_k + A_{k-1} + \ldots + A_{k-n+1}}{n} = \frac{1}{n}\sum_{i=1}^{n} A_{k-i+1}, \quad k = n+1, \ldots, M \tag{4.1}$$

Thus, $\mathscr{M}_{k+1}$ is the average of the previous n values $A_k, \ldots, A_{k-n+1}$, that compose the "data window" of width n. In the applications of MA, often this width n is randomly selected from the set $\{2,3,4\}$. Now, let us consider that A_k is the accuracy, defined by eq. 3.6 in chapter 3, achieved by ELM on the training set using H_k hidden neurons, for $k = 1, \ldots, M$, being $\{H_k\}_{k=1}^M$ be a set of M values of the number H of hidden neurons verifying $H_{k+1} - H_k = \Delta H$ independent on k, so that $H_k = H_1 + (k-1)\Delta H$ for $k = 1, \ldots, M$, i.e., the values in $\{H_k\}_{k=1}^M$ are equally spaced. The MA-ELM network uses the moving average to estimate the accuracy A_{n+1} that ELM is expected to achieve with H_{n+1} hidden neurons. This estimation of eq. 4.1, denoted as A'_{n+1}, uses the n previous accuracies $\{A_{k-i+1}\}_{i=1}^n$ achieved by ELM with $\{H_{k-i+1}\}_{k=1}^n$ hidden neurons, so that $A'_{n+1} = \mathscr{M}_{n+1}$. The A'_{n+1} is compared to A_{n+1}, that is the true accuracy calculated by training the ELM using H_{n+1}. A difference $|A'_{n+1} - A_{n+1}|$ above a threshold τ is considered high, so that MA-ELM uses the true value A_{n+1}. In this case, for $n+2$ both the true A_{n+2} and estimated A'_{n+2} accuracies are calculated, what requires to train again the ELM for H_{n+2}, and the difference test is repeated. On the contrary, when $|A'_{n+1} - A_{n+1}| \leq \tau$, the difference is considered to be low and MA-ELM accepts the value estimated by moving average, so that $A_k = A'_k = \mathscr{M}_k$ for $k = n+2, \ldots, M$. In our case, since A_k are accuracies (in %), the value of the threshold τ is set to a low value τ=0.5. When all the accuracies $\{A_k\}_{k=1}^M$ are available, either estimated using moving average (eq. 4.1) or calculated by ELM training, the selected value H^* for the test is the one with the highest accuracy:

$$H^* = H_{k^*}, \quad k^* = \underset{k=1,\ldots,M}{\arg\max}\,\{A_k\} \tag{4.2}$$

The operation of MA-ELM is compiled by algorithm 6. In this code, function ELM1 is defined in algorithm 1 of chapter 1 and $\delta(x)$ is defined in eq. 3.6 of chapter 3. The function ACC calculates the accuracy of ELM trained using the corresponding H argument. The notation $t_n, z_n \leftarrow \underset{c=1,\ldots,C}{\arg\max}\{t_{cn}, v_{cn}\}$ means that t_n (resp. z_n) is the argument that maximizes t_{cn} (resp.

v_{cn}) over $c = 1,\ldots,C$. The notation $\mathscr{M}_k|\mathscr{E}_k$ in lines 10 and 16 means that function $\mathscr{M}_k$ given by eq. 4.1 (resp. function $\mathscr{E}_k$ in eq. 4.3 below) is used for MA-ELM (resp. for EMA-ELM). It is expected that for many H_k values the A_k is estimated using MA, what saves the corresponding ELM trainings, and MA-ELM would be faster than ELM. However, if the difference test is never true all the A_k values will be calculated training ELM instead of using moving average. In this case, no ELM training would be saved, and MA-ELM would be slightly slower due to the MA calculation.

Algorithm 6: Algorithm for MA-ELM and EMA-ELM. See text for details.

```
1  Algorithm: $[\mathbf{A},\mathbf{B}]$=MA|EMA-ELM($\mathbf{X}$,$\mathbf{T}$,$\{H_k\}_{k=1}^{M}$,$g$)
   Data: $\mathbf{X} = \{x_{ni}\}_{ni=1}^{N,I}$: training set; $\mathbf{T} = \{t_{cn}\}_{cn=1}^{C,N}$: true output; $\{H_k\}_{k=1}^{M}$: set of
         numbers of hidden neurons; $g$: activation function
   Result: $\mathbf{A} = \{a_{hi}\}_{hi=1}^{H,I}$: input weights; $\mathbf{B} = \{b_{ch}\}_{ch=1}^{C,H}$: output weights.
2  Select randomly $n \in \{2,3,4\}$; $\tau \leftarrow 0.5$; train ←True.
3  def ACC($\mathbf{X},\mathbf{T},H,g$):
4      $[\mathbf{A},\mathbf{B}] \leftarrow$ELM1($\mathbf{X},\mathbf{T},H,g$). $\mathbf{V} = \{v_{cn}\}_{cn=1}^{C,N} \leftarrow \mathbf{B}g(\mathbf{A}\mathbf{X}^T)$.
5      $\left\{ t_n, z_n \leftarrow \arg\max_{c=1,\ldots,C}\{t_{cn}, v_{cn}\} \right\}_{n=1}^{N}$; $A \leftarrow \frac{100}{N}\sum_{i=1}^{N}\delta(t_n,z_n)$.
6      return $A$.
7  for $k \leftarrow 1,M$ do
8      if train then
9          $A_k \leftarrow$ACC($\mathbf{X},\mathbf{T},H_k,g$).
10     end
11     if $k > n$ then
12         $A'_k \leftarrow \mathscr{M}_k|\mathscr{E}_k$.
13         if train and $|A_k - A'_k| < \tau$ then
14             $A_k \leftarrow A'_k$; train ←False.
15         end
16     end
17 end
18 $k^* \leftarrow \arg\max_{k=1,\ldots,M}\{A_k\}$; $H^* \leftarrow H_{k^*}$; $[\mathbf{A},\mathbf{B}] \leftarrow$ELM1($\mathbf{X},\mathbf{T},H^*,g$).
```

Table 4.1: Values of γ and the weights of the current and previous samples for each value of n.

n	γ	$1-\gamma$	$\gamma/(n-1)$
2	0.67	0.33	0.67
3	0.5	0.5	0.25
4	0.4	0.6	0.13

4.2 Exponential moving average extreme learning machine

Exponential moving average (EMA) is a type of MA that gives more importance to the expected values, in our case the ELM accuracies, being used to see the direction of the values with the change of the influencing factor, in our case the number H of hidden neurons. This technique has wide uses in the field of artificial intelligence and deep learning [34]. The EMA differs from MA in the mode of calculation, because it gives more weight to the most recent data A_k using a weighting multiplier γ that modulates the sensitivity with respect to changes in their values. The n-width exponential moving average of A_k, denoted as $\{\mathscr{E}_{k+1}$, is defined by eq. 4.3:

$$\mathscr{E}_k = (1-\gamma)A_k + \frac{\gamma}{n-1}\sum_{i=2}^{n} A_{k-i+1}, \quad \gamma = \frac{2}{1+n} \tag{4.3}$$

where $n \in \{2,3,4\}$ as in MA. In EMA, the weight γ of A_k is higher than the weights of the previous samples $\frac{\gamma}{n-1}$, as shown by Table 4.1.

4.3 Divide-and-conquer extreme learning machine

In artificial intelligence, divide-and-conquer (DC) is a strategy for the design of many algorithms and a good tool for solving problem. This method is characterized by dividing a problem into two or more related sub-problems that are simple enough to be directly solved. This strategy is applied for example to solve kernel support vector machine by clustering data [45] in order to divide them into smaller problems. In the context of ELM, [28] showed a fast divide-and-conquer approximation for big data, where the dataset is first sliced into multiple small subsets without overlapping according to a uniform distribution, then each subset generates one part of the hidden neurons for fast-SVD hidden-neurons ELM.

In this chapter of the current thesis we used the divide-and-conquer strategy to determine the best number H^* of hidden neurons in ELM. The proposed method, named divide-and-conquer extreme learning machine (DC-ELM), first divides the set $\mathscr{H} = \{H_k\}_{k=1}^{M}$ of numbers of hidden neurons in two disjoint subsets $\mathscr{H}_1$ and $\mathscr{H}_2$ so that $\mathscr{H} = \mathscr{H}_1 \cup \mathscr{H}_2$. Specifically, if $\mathscr{H} = \{H_k\}_{k=1}^{M}$, a number p is randomly selected from $\{1,\dots,M\}$ and $\mathscr{H}$ is splitted into subsets $\mathscr{H}_1 = \{H_k\}_{k=1}^{p}$ and $\mathscr{H}_2 = \{H_k\}_{k=p+1}^{M}$. Then, CD-ELM proceeds by comparing accuracies achieved by the ELM using the largest H values of $\mathscr{H}_1$ and $\mathscr{H}_1$. The subset that provides the best performance is selected to be subjected again to divide-and-conquer, and the other subset is neglected. The algorithm continues recursively until it reaches a subset with only one number of hidden neurons, that is the one that provides the best accuracy, but with a reduced number of executions of the ELM training. Algorithm 7 compiles this method, where $|\mathscr{H}|$ denotes the cardinal of set $\mathscr{H}$ and function ACC is already defined in algorithm 6.

Algorithm 7: DC-ELM Algorithm. See text for details.

```
1  Algorithm: [$\mathbf{A},\mathbf{B}$]=DC-ELM($\mathbf{X}$,$\mathbf{T}$,$\{H_k\}_{k=1}^{M}$,$g$)
   Data: $\mathbf{X} = \{x_{ni}\}_{ni=1}^{N,I}$: training set; $\mathbf{T} = \{t_{cn}\}_{cn=1}^{C,N}$: true output; $\{H_k\}_{k=1}^{M}$: set of
         numbers of hidden neurons; $g$: activation function
   Result: $\mathbf{A} = \{a_{hi}\}_{hi=1}^{H,I}$: input weights; $\mathbf{B} = \{b_{ch}\}_{ch=1}^{C,H}$: output weights.
2  def DC($\mathscr{H}$):
3      $l \leftarrow |\mathscr{H}|$. $\mathscr{H}^* \leftarrow \mathscr{H}$.
4      if $l > 1$ then
5          $p \leftarrow l/2$. Split $\mathscr{H}$ into subsets $\mathscr{H}_1 = \{H_k\}_{k=1}^{p}$ and $\mathscr{H}_2 = \{H_k\}_{k=p+1}^{l}$.
6          if ACC($\mathbf{X},\mathbf{T},\max\{\mathscr{H}_1\},g$)>ACC($\mathbf{X},\mathbf{T},\max\{\mathscr{H}_2\},g$) then
7              $\mathscr{H}^* \leftarrow \mathscr{H}_1$.
8          else
9              $\mathscr{H}^* \leftarrow \mathscr{H}_2$.
10         end
11     end
12     return $\mathscr{H}^*$.
13 $H^* \leftarrow$DC($\{H_k\}_{k=1}^{M}$); [$\mathbf{A},\mathbf{B}$] $\leftarrow$ELM1($\mathbf{X},\mathbf{T},H^*,g$).
```

Fig. 4.1 shows an example of operation of DC-ELM over a set of $M = 10$ numbers of hidden layer sizes $\mathscr{H} = \{H_k\}_{k=1}^{M} = \{10,20,30,40,50,60,70,80,90,100\}$. A number p=5 is randomly selected from the set $\{1,\dots,10\}$ and $\mathscr{H}$ is splitted in two subsets $\mathscr{H}_1$=$\{$10, 20, 30, 40, 50$\}$ and $\mathscr{H}_2$=$\{$60, 70, 80, 90, 100$\}$. The accuracy of the ELM using the largest value of each subset is evaluated by function ACC in algorithm 6. The best performing size from both subsets is selected and accordingly the subset that contains that size. In the example of Fig. 4.1, the ELM performs better for H=100 (the largest value in $\mathscr{H}_2$, accuracy=83%) than for

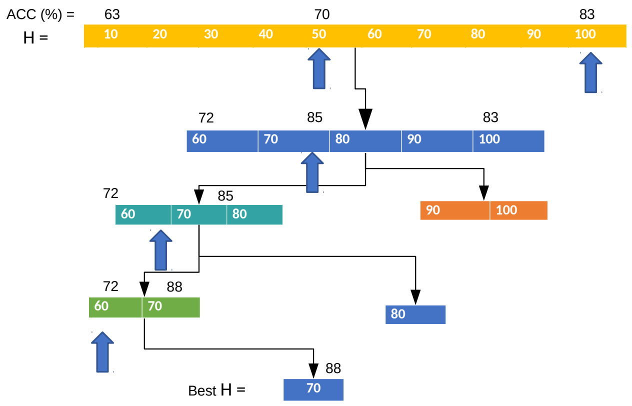


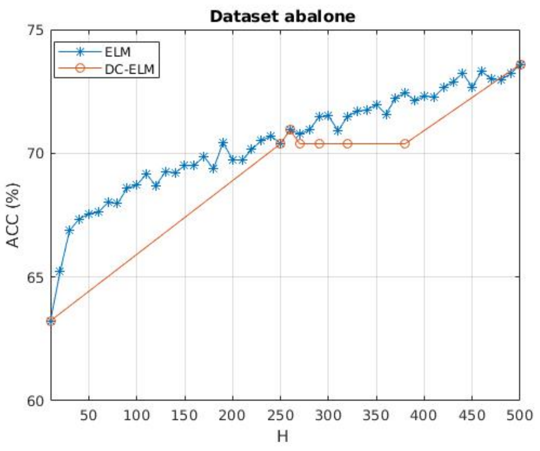
Figure 4.1: Example of the DC-ELM operation.

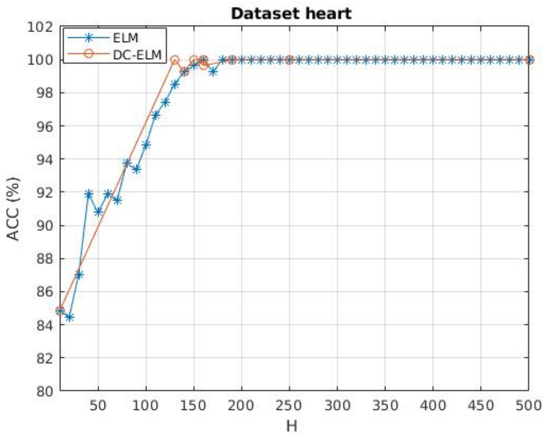
H=50 (the largest value in $\mathscr{H}_1$, accuracy=70%), so the subset $\mathscr{H}_2 = \{60, 70, 80, 90, 100\}$ is selected, and $\mathscr{H} = \mathscr{H}_2$ for the next iteration. Based on that, the divide-and-conquer method continues: now M=5, the value p is randomly set to p=3, the new $\mathscr{H}$ is splitted into two new parts $\mathscr{H}_1 = \{60, 70, 80\}$ and $\mathscr{H}_2 = \{90, 100\}$, and the accuracies of their largest values H=80 and H=100 are calculated. Of course, some of the required accuracies may be already calculated in previous iterations, as for H=100 in this example. The accuracies of the hidden sizes H=100 and H=80 are compared (see Fig. 4.1), and since the latter provides the highest accuracy (85% for H=80 and 83% for H=100), the subset $\mathscr{H}_1 = \{60, 70, 80\}$ is now selected. The method proceeds with repeated trials until a subset of size one is achieved, that in the example of Fig. 4.1 (lower end) is $H^* = 70$. Fig. 4.2 plots the training accuracy of ELM and DC-ELM, showing one marker for each training execution, where we can see that the final performance is equal for both methods, but DC-ELM highly reduces the number of trainings with respect to ELM.

4.4 Experimental methodology

The models MA-ELM, EMA-ELM and DC-ELM were compared with the original ELM and the constrained extreme learning machine, denoted as CELM [134], whose code was downloaded from GitHub site [1]. We used a collection of 27 classification benchmark datasets

[1]https://github.com/wentaozhu/constrained-extreme-learning-machine





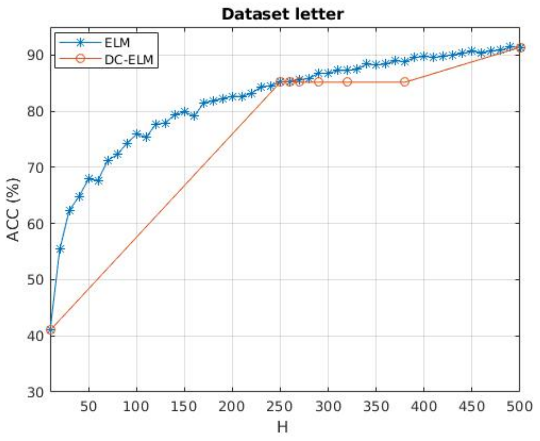


Figure 4.2: Accuracy vs. H during ELM and DC-ELM training for several datasets.

Table 4.2: Collection of UCI classification datasets.

Original Name	Dataset	N	I
Abalone	abalone	4,177	8
Australian Sign Language signs	australian	6,650	15
Chess(King-Rook vs. King)	chess	18,056	6
Congressional Voting Records	voting	435	16
Connect-4	connect-4	67,557	42
Connectionist Bench	sonar	208	60
Energy efficiency	energy-heat	768	8
Hepatitis C Virus (HCV)	hepatitis	1,385	29
Image Segmentation	imseg	2,310	19
Ionosphere	ionosphere	351	34
Letter Recognition	letter	20,000	16
MAGICGamma Telescope	magic	19,020	11
MammographicMass	mammograph	961	6
MiniBooNE	miniboone	130065	50
MicroMass	msa	931	1300
MONK's Problems	monks2	432	7
Nursery	nursery	12,960	8
Pima Indians Diabetes data	pima	768	8
Planning Relax	planning	182	13
Seeds	seeds	210	7
Shuttle Landing Control	shuttle	57,977	6
South German Credit	german	1,000	21
SPECT Heart	heart	267	22
Statlog (Vehicle Silhouettes)	vehicle	946	18
Synthetic Control Chart	synthetic	600	60
Tic-Tac-Toe Endgame	tictac	958	9
Wine	wine	178	13

selected from the UCI Machine Learning Repository [2], whose specifications (numbers N and I of patterns and inputs, respectively) are listed in Table 4.2.

The experiments were executed with a computer equipped with 8 Intel Core i7-4790k processors at 4GHz, having 16 GB RAM and Ubuntu 18.04 operative system. The proposed methods MA-ELM, EMA-ELM and DC-ELM were codified under Matlab R2018. Each

[2]https://archive.ics.uci.edu (Visited May, 2021)

dataset was splited into training, validation, and testing sets, using standard 4-fold cross-validation to evaluate each method. The performance measurement was the accuracy (ACC), in %, already defined in eq. 3.6 of chapter 3. Furthermore, to comprehensively compare the resulting performances, each algorithm used in the experiments is uniformly assigned a number H of hidden neurons varying from H=10 to H=500 with a step of size 10, so that $\mathscr{H} = \{10(k+1)\}_{k=1}^{50}$ and $M = 50$. Given that the ELM must be trained for all the H-values, the number of training executions is M=50. With respect to our proposals, the number M' of training executions, that is expected to be lower than M, depends on the iterations required to choose the appropriate size. Thus, the percentage of reduction in the number of hidden sizes tried, denoted as α (in %), of MA-ELM, EMA-ELM or DC-ELM with respect to the original ELM is given by eq. 4.4:

$$\alpha(\%) = \frac{100(M-M')}{M} \tag{4.4}$$

The percentage of reduction in time, denoted as β (also in %) of MA-ELM, EMA-ELM and DC-ELM with respect to the original ELM and CELM is given by in eq 4.5:

$$\beta(\%) = \frac{100(T-T')}{T} \tag{4.5}$$

where T and T', that is expected to be lower than T, are the times spent by ELM or CELM and by our approaches, respectively.

4.5 Comparison of MA-ELM with ELM and CELM

Table 4.3 reports the accuracies achieved by MA-ELM and ELM. The proposed method EM-ELM achieves a time reduction β up to 94.4% with respect to the original ELM. The ratio between the times of ELM and MA-ELM is 9.11 which means that MA-ELM is, on average, nine times faster than the original ELM achieving the same average accuracy (79.9%). For instance, in `seeds` dataset α=94.4% and MA-ELM achieves a higher accuracy (95.3%) than ELM (93.9%), while the times are 0.12 and 2.15 s., respectively, with a time reduction percentage β=96.2%. This means that, in average, MA-ELM performed the H tuning in 3.8% of the time spent by ELM, with similar or better results as shown by the average accuracy. Note that, from eq. 4.5:

$$\frac{T}{T'} = \frac{100}{100-\beta} \tag{4.6}$$

so that a value β=98%, such as in dataset `wine`, means that ELM is about 50 times slower than MA-ELM, because $T/T' = 100/(100-98) = 100/2 = 50$.

We also combined the moving average to the constrained, constrained sum and constrained difference ELM (CELM, CSELM and DELM, respectively), that provide alternatives to randomly select numbers of hidden neurons [134]. Table 4.4 reports the results of CELM, CSELM and DELM compared to their MA- counterparts MA-CELM, MA-DSELM and MA-DELM, respectively. The average accuracy of MA-CELM (80.8%) outperforms CELM (80.3%) with average β up to 92.1%. Analogously, MA-CSELM and MA-DELM achieve 81.1% and 80.5% outperforming CSELM and DELM (81% and 80.4%, respectively), with average β up to 83.8% and 91.7%. Moreover, in `mammograph` dataset the CELM achieved ACC=66.3% with β=93.7%, while EMA-CELM achieved ACC=78.6%. With respect to MA-CSELM, in the `australian` dataset achieved ACC=84.9%, were CSELM achieved ACC=76.5% with β=90.4%.

The MA was also combined with mixed, sum and random sum ELM (MELM, RSELM and SELM, respectively [134]), and the comparison to the original methods (i.e., without MA) is reported in Table 4.5. The MA-SELM achieved an average ACC=80.5% outperforming SELM (ACC=79.9%) with a high average β=94.3%. Besides, MA-RSELM achieved avg. ACC=80.4% while RSELM achieved 80.2% with β up to 94.8%. Moreover, in MA-MELM the avg. ACC=81.0% while MELM achieved 80.6% with average β=87.5%

4.6 Comparison of EMA-ELM with ELM and CELM

In this subsection the proposed EMA-ELM is compared with original ELM. As in the case of moving average, the conducted experiments were performed by randomly choosing the window width n randomly in the set $\{2,3,4\}$. The results showed a significant improvement in the percentage of reduction in the number of hidden sizes tried (α), in the time (T') and in the percentage of reduction in time (β), while achieving similar or higher accuracy, as reported in Table 4.6. The EMA-ELM achieved slightly better ACC=80.0% than the original ELM (79.9%) and with β ranging between 60% and 96%. The time ratio (bottom of the table) reports that EMA-ELM is in average 7.4 faster than than ELM. In addition, EMA-ELM proved

Table 4.3: Comparison of MA-ELM and ELM.

	ELM			MA-ELM					
Dataset	ACC	H	T(s)	ACC	H	T'(s)	M	α(%)	β(%)
abalone	**66.2**	500	29.27	64.6	60	2.41	6	91.8	88
australian	76.2	500	10.78	**84.9**	40	0.8	4	92.6	92
chess	**42.4**	490	304.6	30.7	100	243.75	10	20.0	80
connect-4	**75.5**	500	1296.25	68.1	30	119.37	3	90.8	94
energy-heat	**92.8**	390	7.83	88.7	90	0.82	9	89.5	82
german	65.3	240	6.69	**71.9**	110	1.04	11	84.5	78
heart	76.7	160	3.84	**80.1**	40	0.4	4	89.6	92
hepatitis	78.7	90	2.27	**81.3**	70	0.4	7	82.4	86
imseg	**94.7**	460	19.36	89.9	110	2.85	11	85.3	78
ionosphere	83.7	200	5.66	**88.0**	120	0.89	12	84.3	76
letter	**77.0**	490	157.68	71.9	200	51.54	20	67.3	60
magic	**84.7**	500	130.87	79.3	60	13.8	6	89.5	88
mammograph	68.4	370	6.99	**77.2**	40	0.51	4	92.7	92
miniboone	**90.7**	490	1799.25	86.3	60	228.57	6	87.3	88
monks2	52.9	280	5.07	**63.4**	160	1.34	16	73.6	68
msa	**98.8**	30	7.41	**98.8**	30	0.55	3	92.6	94
nursery	**92.6**	470	108.55	88.3	50	10.8	5	90.1	90
pima	66.8	500	7.53	**76.0**	50	0.55	5	92.7	90
planning	**52.4**	130	2.13	**52.4**	130	0.49	13	77.0	74
seeds	93.9	80	2.15	**95.3**	20	0.12	2	94.4	96
shuttle	**96.8**	410	406.30	90.5	80	41.31	8	89.8	84
sonar	**72.6**	130	4.35	72.2	90	0.66	9	84.8	82
synthetic	**94.8**	170	11.76	93.2	80	1.73	8	85.3	84
tictac	**97.8**	320	10.58	95.4	80	1.47	8	86.1	84
vehicle	**80.1**	480	9.55	79.3	130	1.55	13	83.8	74
voting	90.9	150	5.35	**93.3**	50	0.51	5	90.5	90
wine	**95.0**	10	2.34	**95.0**	10	0.12	2	94.9	98
Avg.	**79.9**			**79.9**		9.11		84.6	96.2

that hidden size do not have to be the largest in order to achieve better performance in some case. For example, in the `australian` dataset achieved in original ELM best H=500 with ACC=76.2% and T=10.78 sec., where the EMA-ELM selected H=70 with ACC=85.4% and T'=1.19 s., α= 89% and β=86%. In the the `synthetic` dataset, the original ELM achieved

Table 4.4: Accuracy and β of CELM, CSELM and DELM with and without MA.

	CELM	MA-CELM		CSELM	MA-CSELM		DELM	MA-DELM	
Dataset	ACC	ACC	β	ACC	ACC	β	ACC	ACC	β
abalone	**66.6**	64.7	92.8	**67.1**	65.4	87.8	**66.5**	64.3	94.4
australian	78.1	**85.4**	94.7	76.5	**84.9**	90.4	77.1	**85.5**	93.5
chess	**43.4**	30.7	87.2	**49.2**	47.8	11.1	**43.2**	31.2	86.1
connect-4	**76.7**	73.8	85.9	**77.4**	73.3	82.1	**76.6**	73.9	86.1
energy-heat	**94.4**	88.1	94.6	**93.7**	92.7	83.1	**94.3**	88.1	93.4
german	60.3	**71.6**	91.3	64.8	**69.2**	84.4	65.9	**70.0**	92.3
heart	79.0	**82.0**	94.7	73.4	**78.6**	89.7	76.3	**81.5**	92.6
hepatitis	79.3	**86.5**	94.2	**81.3**	**81.3**	85.0	76.1	**82.6**	91.6
imseg	**95.4**	87.8	94.3	**95.0**	90.5	86.9	**95.4**	88.0	93.2
ionosphere	**85.7**	83.7	94.6	82.3	**86.0**	83.6	84.3	**84.5**	93.1
letter	**75.7**	65.2	85.8	**74.2**	69.7	71.3	**75.7**	65.8	84.3
magic	**84.5**	80.0	94.8	**84.8**	80.8	90.1	**84.5**	80.3	93.5
mammograph	66.3	**78.6**	93.7	73.3	**77.4**	84.7	74.1	**76.9**	94.9
miniboone	**90.3**	86.8	91.8	**90.5**	83.1	90.6	**90.3**	84.9	93.3
monks2	52.2	**60.2**	92.8	49.8	**59.7**	89.1	53.9	**64.4**	94.1
msa	99.4	**100**	94.9	**99.5**	**99.5**	93.7	99.7	**100**	95.4
nursery	**94.3**	88.9	94.0	**95.5**	90.5	86.3	**94.3**	89.3	93.0
pima	69.3	**76.2**	95.2	66.8	**76.6**	94.4	66.4	**75.3**	94.3
planning	50.5	**64.7**	88.4	**54.5**	**54.5**	78.5	49.3	**62.5**	84.8
seeds	92.9	**93.9**	93.9	93.4	**93.8**	93.1	93.9	**94.8**	94.6
shuttle	**98.6**	94.3	94.0	**97.9**	91.5	88.7	**98.3**	93.5	95.2
sonar	75.2	**75.9**	86.8	**81.4**	**81.4**	80.3	**76.1**	75.4	89.5
synthetic	**97.2**	95.3	88.8	**96.7**	95.3	88.0	**97.5**	94.0	90.5
tictac	**98.5**	97.5	89.9	**98.5**	97.6	85.0	**98.8**	97.9	89.2
vehicle	**80.0**	76.2	87.6	**80.8**	77.3	85.7	**80.5**	75.3	87.8
voting	85.8	**94.2**	94.5	92.7	**94.5**	90.4	84.5	**94.5**	94.4
wine	97.9	**98.6**	95.1	**97.2**	**97.2**	93.6	**98.6**	**98.6**	94.0
Avg.	80.3	**80.8**	92.1	81.0	**81.1**	83.8	80.4	**80.5**	91.7

ACC=94.8% and T=11.76 s. with H=170 hidden neurons, while EMA-ELM achieved better ACC=95.3% with H=120, T'=2.62 s., α=77.7% and β=76%.

Analogously to MA-ELM, Table 4.7 reports the comparison of EMA-CELM, EMA-CSELM and EMA-DELM with CELM, CSELM and DELM, respectively. These results prove that

Table 4.5: Accuracy and β of MELM, RSELM and SELM with and without MA.

	MELM	MA-MELM		RSELM	MA-RSELM		SELM	MA-SELM	
Dataset	ACC	ACC	β	ACC	ACC	β	ACC	ACC	β
abalone	**66.9**	65.2	90.6	**66.4**	64.2	95.4	**66.3**	66.0	90.8
australian	74.3	**85.5**	91.2	73.9	**85.7**	96.2	77.2	**85.7**	89.4
chess	**46.1**	36.6	68.4	**43.3**	30.7	87.7	**49.3**	48.6	05.2
connect-4	**77.2**	73.3	80.8	**76.9**	74.2	86.3	**77.5**	72.7	82.8
energy-heat	**94.1**	88.7	86.1	**94.9**	84.4	94.4	**95.2**	89.1	82.9
german	62.0	**69.4**	89.1	58.7	**70.5**	96.2	62.0	**67.2**	84.5
heart	74.1	**84.1**	94.2	77.8	**82.6**	98.1	71.9	**79.7**	82.7
hepatitis	85.2	**85.2**	87.7	80.6	**86.5**	96.5	**83.3**	**83.3**	81.2
imseg	**95.4**	90.9	84.0	**95.5**	89.0	94.0	**94.9**	90.2	82.3
ionosphere	82.5	**86.4**	89.1	84.6	**85.3**	95.0	85.1	**85.3**	76.4
letter	**75.4**	71.0	69.3	**71.9**	63.7	86.5	**74.0**	70.7	62.7
magic	**84.6**	81.1	90.7	**84.0**	77.9	93.7	**84.8**	81.0	88.4
mammograph	71.0	**78.6**	90.4	68.2	**77.1**	97.0	69.3	**77.1**	93.1
miniboone	**90.5**	87.5	90.7	**89.9**	83.0	93.5	**90.5**	88.1	87.3
monks2	46.5	**62.0**	85.6	53.0	**65.1**	98.1	49.0	**61.1**	86.6
msa	**98.7**	**98.7**	94.3	99.7	**99.8**	97.4	**100**	**100**	93.1
nursery	**95.4**	89.7	91.4	**94.2**	89.2	94.5	**95.2**	89.3	91.5
pima	68.5	**75.5**	93.9	68.4	**75.7**	97.2	67.3	**77.3**	80.1
planning	**48.8**	**48.8**	77.5	59.3	**68.1**	98.0	**56.7**	**56.7**	75.0
seeds	**93.9**	93.8	91.7	93.8	**95.8**	98.4	93.8	**94.3**	91.7
shuttle	**98.7**	94.2	93.1	**99.0**	91.1	95.1	**97.9**	91.6	88.8
sonar	78.0	**78.4**	78.5	**74.6**	74.4	97.1	**80.4**	**80.4**	78.8
synthetic	**97.2**	96.0	88.0	**94.2**	90.0	91.6	**96.2**	94.5	88.0
tictac	**98.6**	97.3	86.8	**98.5**	98.3	96.3	98.3	**98.4**	83.0
vehicle	**79.5**	**78.2**	82.3	**79.6**	75.3	94.3	**80.6**	76.4	86.2
voting	**94.2**	93.9	92.2	86.0	**94.2**	97.7	—	—	—
wine	**97.8**	**97.8**	93.5	**98.6**	**98.6**	98.4	**98.6**	**98.6**	93.2
Avg	80.6	**81.0**	87.5	80.2	**80.4**	94.8	79.9	**80.5**	94.3

EMA-CELM achieves average β=95.1% and average ACC=80.4%, while CELM achives avg. ACC=80.3%. The comparison of EMA-CSELM and CSELM leads to average β=92.6% and avg. ACC=81.2$ for EMA-CSELM outperforming CSELM (81.0%). Moreover, in the `monks2` dataset the DELM achieves ACC=53.9%, while EMA-DELM achieves ACC=55.3%

Table 4.6: Comparison of ELM and EMA-ELM.

	ELM			EMA-ELM					
Dataset	ACC	H	T(s)	ACC	H	T'(s)	M	α(%)	β(%)
abalone	**66.2**	500	29.27	64.6	70	3.52	7	88.0	86
australian	76.2	500	10.78	**85.4**	70	1.19	7	89.0	86
chess	**42.4**	490	304.6	34.9	200	118.69	20	61.0	60
connect-4	**75.5**	500	1296.25	68.1	30	119.69	3	90.8	94
energy-heat	**92.8**	390	7.83	88.7	90	0.89	9	88.6	82
german	65.3	240	6.69	**68.3**	140	1.7	14	74.6	72
heart	**76.7**	160	3.84	**76.7**	160	1.06	16	72.4	68
hepatitis	**78.7**	90	2.27	**78.7**	90	0.45	9	80.2	82
imseg	**94.7**	460	19.36	91.8	140	4.29	14	77.8	72
ionosphere	83.7	200	5.66	**84.5**	110	0.98	11	82.7	78
letter	**77.0**	490	157.68	68.5	150	40.16	15	74.5	70
magic	**84.6**	500	130.87	81.2	90	19.01	9	85.5	82
mammograph	68.4	370	6.99	**78.5**	170	1.73	17	75.3	66
miniboone	**90.7**	490	1799.25	88.5	100	359.54	10	80.0	80
monks2	52.9	280	5.07	**64.1**	20	0.21	2	95.9	96
msa	**98.8**	30	7.41	**98.8**	30	0.7	3	90.6	94
nursery	**92.6**	470	108.55	89.2	100	18.55	10	82.9	80
pima	66.8	500	7.53	**76.8**	90	1.07	9	85.8	82
planning	**52.4**	130	2.13	**52.4**	130	0.79	13	62.9	74
seeds	93.9	80	2.15	**95.3**	20	0.17	2	92.1	96
shuttle	96.8	410	406.30	**90.8**	90	57.57	9	85.8	82
sonar	**72.6**	130	4.35	**72.6**	130	1.26	13	71.0	74
synthetic	94.8	170	11.76	**95.3**	120	2.62	12	77.7	76
tictac	**97.8**	320	10.58	97.5	140	2.11	14	80.1	72
vehicle	**80.1**	480	9.55	79.6	150	2.24	15	76.5	70
voting	90.9	150	5.35	**93.3**	50	0.47	5	91.2	90
wine	**95.0**	10	2.34	**95.0**	10	0.12	1	94.9	98
Avg.	79.9			**80.0**		7.4		81.8	80.1

with β=79.5%, and EMA-DELM achieves average β=96%. Note that the percentage β of reduction in time is very low in dataset `chess` because the condition $|A'_k - A_k| < \tau$ on algorithm 6 is rarely fulfilled, so the ELM must be trained for almost all the H values.

The results in Table 4.8 compare SELM, CELM and CSELM to their versions combined

Table 4.7: Accuracy and β of CELM, CSELM and DELM with and without EMA.

	CELM	EMA-CELM		CSELM	EMA-CSELM		DELM	EMA-DELM	
Dataset	ACC	ACC	β	ACC	ACC	β	ACC	ACC	β
abalone	**66.6**	65.6	84.5	**67.1**	65.6	82.0	**66.5**	65.1	88.1
australian	78.1	**85.2**	78.8	76.5	**84.1**	71.2	77.1	**85.2**	79.3
chess	**43.4**	41.9	9.5	**49.2**	**49.2**	1.1	**43.2**	**43.2**	1.8
connect-4	**76.7**	74.4	76.9	**77.4**	75.2	63.3	**76.6**	74.2	79.4
energy-heat	**94.4**	89.7	77.8	**93.7**	92.7	74.3	**94.3**	88.5	80.2
german	60.3	**64.8**	66.4	**64.8**	**64.8**	52.3	65.9	**67.5**	60.8
heart	**79.0**	**79.0**	72.1	**73.4**	**73.4**	69.5	**76.3**	**76.3**	69.5
hepatitis	**79.3**	**79.3**	80.8	**81.3**	**81.3**	78.5	**76.1**	**76.1**	82.8
imseg	**95.4**	92.2	76.9	95.0	**92.5**	79.4	**95.4**	92.0	77.0
ionosphere	**85.7**	85.4	81.1	**82.3**	**82.3**	73.8	84.3	**86.0**	76.4
letter	**75.7**	73.6	34.5	**74.2**	72.9	29.0	**75.7**	73.1	45.3
magic	**84.5**	81.4	85.5	**84.8**	81.8	86.2	**84.5**	81.1	88.6
mammograph	66.3	**78.3**	84.2	73.3	**75.7**	59.8	74.1	**76.9**	95.6
miniboone	**90.3**	87.6	85.2	**90.5**	88.5	79.6	**90.3**	87.1	86.0
monks2	**52.2**	49.5	72.2	49.8	**56.9**	81.0	53.9	**55.3**	79.5
msa	**99.4**	**99.4**	93.0	**99.5**	**99.5**	89.4	**99.7**	**99.7**	93.8
nursery	**94.3**	88.9	87.0	**95.5**	93.5	66.8	**94.3**	89.4	87.8
pima	69.3	**75.8**	95.1	66.8	**72.9**	55.5	66.4	**76.8**	89.3
planning	**50.5**	**50.5**	70.9	**54.5**	**54.5**	69.5	**49.3**	**49.3**	74.3
seeds	92.9	**93.3**	90.2	**93.4**	**93.4**	86.9	**93.9**	92.9	90.7
shuttle	**98.6**	94.8	91.2	**97.9**	92.1	83.3	**98.3**	95.6	91.7
sonar	**75.2**	**75.2**	73.8	**81.4**	**81.4**	73.4	**76.1**	**76.1**	73.6
synthetic	**97.2**	95.3	85.1	**96.7**	96.0	82.3	**97.5**	95.8	84.0
tictac	**98.5**	98.0	84.7	**98.5**	98.3	75.0	**98.8**	98.0	85.6
vehicle	80.0	**80.5**	63.4	80.8	**82.0**	68.8	80.5	**80.5**	64.1
voting	85.8	**94.2**	85.2	92.7	**94.5**	85.8	84.5	**93.9**	78.8
wine	**97.9**	**97.9**	92.8	**97.2**	**97.2**	92.6	**98.6**	**98.6**	93.2
Avg.	80.3	**80.4**	95.1	81.0	**81.2**	92.6	80.4	**80.5**	96.0

with EMA. The EMA-SELM achieved average β=93.1% with ACC=80.2% outperforming SELM (79.9%). As well, in `monks2` dataset SELM achieved ACC=49% while EMA-SELM achieves 54.8% with β=71.7%. With respect to EMA-RSELM and RSELM, the average *beta* reaches 96.7% with ACC=80.5% compared to 80.2% with RSELM. In the `voting`

Table 4.8: Accuracy and β of MELM, RSELM and SELM with and without EMA.

	MELM	EMA-MELM		RSELM	EMA-RSELM		SELM	EMA-SELM	
Dataset	ACC	ACC	β	ACC	ACC	β	ACC	ACC	β
abalone	**66.9**	65.6	82.3	**66.4**	65.0	87.4	**66.3**	65.0	87.2
australian	74.3	**84.9**	94.5	73.9	**83.5**	78.4	77.2	**85.7**	87.0
chess	**46.1**	**46.1**	-5.6	**43.3**	**43.3**	-0.4	**49.3**	**49.3**	-4.1
connect-4	**77.2**	74.1	77.3	**76.9**	74.8	74.5	**77.5**	74.1	75.6
energy-heat	**94.1**	86.5	95.4	**94.9**	90.9	86.7	**95.2**	91.1	76.8
german	62.0	**70.8**	86.3	**58.7**	**58.7**	64.2	62.0	**64.8**	55.3
heart	74.1	**78.2**	81.1	**77.8**	**77.8**	82.3	71.9	**74.5**	66.3
hepatitis	**85.2**	**85.2**	80.2	**80.6**	**80.6**	92.7	83.3	**83.9**	92.4
imseg	**95.4**	91.5	77.0	**95.5**	92.5	79.4	**94.9**	91.6	77.2
ionosphere	82.5	**83.0**	76.1	84.6	**85.4**	85.9	85.1	**85.3**	74.2
letter	**75.4**	72.7	42.6	**71.9**	71.8	00.3	**74.0**	71.1	45.2
magic	**84.6**	81.8	85.2	**84.0**	80.6	82.1	**84.8**	81.0	87.9
mammograph	71.0	**78.6**	90.4	68.2	**77.4**	73.7	69.3	**77.1**	93.1
miniboone	**90.5**	87.7	84.5	**89.9**	86.7	76.3	**90.5**	87.6	84.0
monks2	46.5	**54.6**	79.3	**53.0**	**53.0**	53.8	49.0	**54.8**	71.7
msa	**98.7**	**98.7**	93.1	**99.7**	**99.7**	93.0	**100**	**100**	92.2
nursery	**95.4**	90.7	82.6	**94.2**	89.5	81.1	95.2	**93.0**	71.1
pima	68.5	**74.7**	95.9	68.4	**75.7**	59.5	67.3	**75.5**	86.8
planning	**48.8**	**48.8**	72.8	**59.3**	**59.3**	87.9	**56.7**	**56.7**	66.2
seeds	**93.9**	93.8	89.8	**93.8**	**93.8**	95.1	93.8	**94.3**	88.6
shuttle	**98.7**	93.6	90.8	**99.0**	93.0	87.8	**97.9**	91.3	87.2
sonar	**78.0**	**78.0**	74.9	**74.6**	**74.6**	87.6	**80.4**	**80.4**	73.8
synthetic	**97.2**	94.7	86.7	**94.2**	92.7	85.5	**96.2**	95.8	83.9
tictac	**98.6**	98.5	79.2	**98.5**	98.3	89.0	**98.3**	97.8	80.7
vehicle	79.5	**80.9**	68.4	79.6	**83.0**	54.2	**80.6**	76.4	85.1
voting	94.2	**94.5**	84.0	86.0	**92.7**	85.6	—	—	—
wine	**97.8**	**97.8**	91.6	**98.6**	**98.6**	97.0	**98.6**	**98.6**	90.5
Avg.	80.6	**81.0**	95.9	80.2	**80.5**	96.7	79.9	**80.2**	93.1

dataset the RSELM achieves ACC=86% while EMA-RSELM reaches 92.7% with β=85.6%. Moreover, comparing EMA-MELM and MELM the average β is 95.9%, while EMA-MELM achieves avg. ACC=81% and MELM ACC=80.6%. In the `pima` dataset the MELM recorded ACC=68.5% while EMA-MELM achieved 74.7% with β=95.9%. Again, β is very slow in dataset `chess` and negative because the accuracy is always calculated by training ELM and never estimated using EMA, so EMA-ELM is slower than ELM.

Table 4.9: Comparison between DC-ELM and ELM.

	ELM			DC-ELM					
Dataset	ACC	H	T(s)	ACC	H	T'(s)	M	α(%)	β(%)
abalone	**66.2**	500	29.27	**66.2**	500	6.22	8	78.8	84
australian	**76.2**	500	10.78	**76.2**	500	2.21	8	79.5	84
chess	**42.4**	490	304.6	42.1	500	59.68	8	80.4	84
connect-4	**75.5**	500	1296.25	**75.5**	500	268.51	8	79.3	84
energy-heat	92.8	390	7.83	**93.0**	500	1.72	8	78.0	84
german	**65.3**	240	6.69	**65.3**	240	1.51	10	77.4	80
heart	**76.7**	160	3.84	**76.7**	160	0.69	8	82.0	84
hepatitis	**78.7**	90	2.27	**78.7**	90	0.45	8	80.2	84
imseg	94.7	460	19.36	**94.9**	500	4.06	8	79.0	84
ionosphere	83.7	200	5.66	**84.9**	210	1.15	9	79.7	82
letter	**77.0**	490	157.68	**77.0**	500	32.23	8	79.6	84
magic	**84.7**	500	130.87	**84.7**	500	29.36	9	77.6	82
mammograph	**68.4**	370	6.99	63.2	500	1.72	8	75.7	84
miniboone	**90.7**	490	1799.25	90.6	500	355.93	8	80.2	84
monks2	**52.9**	280	5.07	56.3	320	1.12	8	77.9	84
msa	98.8	30	7.41	**99.9**	40	1.17	8	84.2	84
nursery	92.6	470	108.55	**92.7**	500	21.78	8	79.9	84
pima	**66.8**	500	7.53	**66.8**	500	1.69	8	77.6	84
planning	**52.4**	130	2.13	**52.7**	130	0.45	9	78.9	82
seeds	**93.9**	80	2.15	**93.9**	90	0.45	8	79.1	84
shuttle	96.8	410	406.30	**97.5**	500	85.04	8	79.1	84
sonar	**72.6**	130	4.35	**72.6**	130	0.87	9	80.0	82
synthetic	**94.8**	170	11.76	**94.8**	210	2.33	9	80.2	82
tictac	**97.8**	320	10.58	**97.8**	320	2.21	8	79.1	84
vehicle	**80.1**	480	9.55	79.8	500	2.18	8	77.2	84
voting	90.9	150	5.35	**91.0**	180	1.06	9	80.2	82
wine	95.0	10	2.34	**96.5**	40	0.44	8	81.2	84
Avg.	79.9			**80.0**		4.9		79.3	83.4

4.7 Comparison of DC-ELM with ELM and CELM

Table 4.10: Accuracy and β of CELM, CSELM and DELM with and without DC.

	CELM	DC-CELM		CSELM	DC-CSELM		DELM	DC-DELM	
Dataset	ACC	ACC	β	ACC	ACC	β	ACC	ACC	β
abalone	**66.6**	**66.6**	75.0	**67.1**	66.4	78.3	66.5	**67.1**	80.4
australian	**78.1**	76.4	79.2	76.5	**79.4**	79.5	77.1	**78.4**	78.2
chess	**43.4**	**43.4**	80.0	**49.2**	49.1	80.7	43.2	**49.2**	81.0
connect-4	**76.7**	**76.7**	80.6	77.4	**77.5**	81.6	76.6	**77.4**	81.5
energy-heat	**94.4**	94.2	77.1	93.7	**94.4**	78.6	**94.3**	93.9	78.2
german	**60.3**	57.6	77.7	**64.8**	61.5	79.5	**65.9**	61.8	80.3
heart	**79.0**	**79.0**	80.3	**73.4**	68.9	77.9	**76.3**	73.4	80.6
hepatitis	**79.3**	**79.3**	79.1	**81.3**	80.6	81.8	76.1	**81.3**	82.8
imseg	95.4	**95.8**	78.0	95.0	**95.4**	80.9	**95.4**	95.0	78.1
ionosphere	**85.7**	**85.7**	81.5	**82.3**	81.7	81.8	**84.3**	82.3	80.7
letter	**75.7**	**75.7**	79.6	**74.2**	74.0	81.0	**75.7**	74.2	79.8
magic	**84.5**	**84.5**	79.7	84.8	**84.8**	78.0	84.5	**84.8**	78.2
mammograph	66.3	**67.9**	74.9	**73.3**	69.4	77.2	74.1	**77.1**	80.2
miniboone	**90.3**	**90.3**	80.0	90.5	**90.5**	80.6	90.3	**90.5**	81.1
monks2	**52.2**	**52.2**	76.3	49.8	**50.7**	76.9	**53.9**	47.5	78.0
msa	99.4	**100**	83.9	99.5	**100**	85.4	99.7	**100**	86.5
nursery	**94.3**	**94.3**	79.3	95.5	**95.7**	80.1	94.3	**95.6**	80.8
pima	**69.3**	**69.3**	76.3	66.8	**67.8**	77.8	66.4	**67.7**	76.6
planning	50.5	**51.6**	77.5	**54.5**	50.5	79.3	49.3	**54.5**	79.3
seeds	92.9	**93.3**	80.7	93.4	**95.8**	83.0	93.9	**94.8**	80.5
shuttle	98.6	**98.8**	78.6	97.9	**98.1**	78.9	**98.3**	98.1	80.3
sonar	**75.2**	**75.2**	79.2	**81.4**	79.8	81.2	76.1	**81.4**	81.2
synthetic	**97.2**	**97.2**	79.5	**96.7**	96.0	79.9	**97.5**	96.7	78.4
tictac	**98.5**	**98.5**	78.5	**98.5**	98.4	80.5	**98.8**	98.5	80.4
vehicle	**80.0**	78.9	75.8	**80.8**	77.7	76.9	**80.5**	79.7	75.1
voting	85.8	**86.9**	75.2	92.7	**93.9**	82.2	84.5	**92.7**	82.3
wine	97.9	**98.6**	80.8	97.2	**98.6**	83.8	98.6	**99.3**	83.1
Avg.	**80.3**	**80.3**	78.7	**81.0**	80.6	80.1	80.4	**81.2**	80.2

Now, we will compare DC-ELM with the original ELM and CELMs. The divide-and-conquer method is used to reach the best-hidden size instead of training all hidden neurons in traditional ways, as discussed in section 4.3. The ELM is only trained using the candidate numbers of hidden neurons, and ignoring the unsuitable numbers of hidden neurons. This reduces the number of training executions in DC-ELM compared to ELM, as shown in Fig 4.2. Table 4.9 reports the accuracy, best H and elapsed time (T) of the original ELM, alongside with the accuracy, best H, time (T'), number M of training executed, percentage α of reduction in M and percentage β of reduction in the training time of DC-ELM for each UCI benchmark classification dataset. The results show that the proposed method DC-ELM achieves α values between 75.4% and 84.2% and β between 82% and 84%. The DC-ELM slightly overcomes ELM in terms of ACC, with average values 80% and 79.9%, respectively. For instance, in the `miniboone` dataset, DC-ELM achieved T'=355.93 sec, while the original ELM achieved T= 1799.25 s. with an ACC (90.6%) close to ELM (90.7%), with α=80.2% and β=84%. Regarding the `shuttle` dataset, DC-ELM achiveds T'=85.04 s. and ACC=97.5% outperforming ELM (T=406.30 s., ACC=96.8%) with α=79.1% and β= 84%. In addition, in the `nursery` dataset, DC-ELM achieved T'=21.78 s. while the ELM spends T=108.55 s., whereas the ACC=92.7% in DC-ELM and 92.6% in ELM, with α=79.9% and β=84%.

The results also showed that there is no significant difference between ELM and DC-ELM in choosing the appropriate H. For example, in the dataset `tictac`, the original ELM selected H=320 with T=10.58 s., which is the same H value selected by DC-ELM with lower T'=2.21 s. and α=79.1% and β=84%. In the `german` dataset, the original ELM selected H=240 and spent T=6.69 s., which is the same H selected by DC-ELM with T'=1.51 s., α=77.4% and β=80%, and both achieved the same ACC=65.3%. This proves that DC-ELM has higher ACC and more speed in choosing the appropriate hidden neuron compared to the original ELM. The time ratio (last line in Table 4.9) between the original ELM and DC-ELM is T/T' =4.87, which means that ELM is about five times slower than DC-ELM.

Similarly to MA-ELM and EMA-ELM, the proposed method DC-ELM was also combined with SELM, CELM and CSELM. The results are reported in Table 4.10, proving that DC-ELM achieves accuracy similar or higher than CSELMs with high β values. The DC-CELM achieves an average β=78.7%, with the same ACC=80.3% as CELM. Comparing CSELM and DC-CSELM, the percentage of training time reduction β ranges between 76.9% to 85.3%. Considering average values, the average ACC is 80.6% and 81% for DC-CSELM and CSELM, respectively. For instance, in `shuttle` dataset the CSELM achieved ACC=97.9% while DC-CSELM achieved ACC=98.1 with β=78.9%. The combination of DC-DELM and DELM, leaded to a β range between from 75.1% and 86.5%, with an average

Table 4.11: Accuracy and β of MELM, RSELM and SELM with and without DC.

	MELM	DC-MELM		RSELM	DC-RSELM		SELM	DC-SELM	
Dataset	ACC	ACC	β	ACC	ACC	β	ACC	ACC	β
abalone	**66.9**	**66.9**	76.6	**66.4**	**66.4**	78.6	**66.3**	66.1	76.8
australian	**74.3**	**74.3**	78.5	**73.9**	**73.9**	77.8	77.2	**77.7**	77.0
chess	**46.1**	**46.1**	79.8	**43.3**	**43.3**	80.1	**49.3**	**49.3**	80.2
connect-4	**77.2**	**77.2**	80.6	**76.9**	**76.9**	81.1	**77.5**	**77.5**	80.0
energy-heat	**94.1**	93.9	76.9	**94.9**	**94.9**	78.6	**95.2**	93.6	76.3
german	62.0	**63.3**	78.2	**58.7**	58.1	79.6	**62.0**	59.0	77.1
heart	**74.1**	70.4	78.1	**77.8**	**77.8**	85.1	**71.9**	68.2	76.2
hepatitis	**85.2**	**85.2**	81.0	**80.6**	**80.6**	88.6	**83.3**	81.3	78.0
imseg	**95.4**	**95.4**	78.5	95.5	**95.6**	75.7	94.9	**95.4**	78.2
ionosphere	**82.5**	81.7	78.6	**84.6**	**84.6**	84.4	**85.1**	**85.1**	78.8
letter	**75.4**	**75.4**	79.6	**71.9**	**71.9**	79.7	**74.0**	**74.0**	79.7
magic	**84.6**	**84.6**	79.2	**84.0**	**84.0**	79.3	**84.8**	**84.8**	79.4
mammograph	**71.0**	**71.0**	77.0	**68.2**	67.9	76.8	69.3	**72.9**	74.6
miniboone	**90.5**	**90.5**	81.2	**89.9**	**89.9**	80.3	**90.5**	**90.5**	79.7
monks2	46.5	**49.3**	76.8	**53.0**	**53.0**	76.4	49.0	**52.6**	75.5
msa	**98.7**	**98.7**	85.2	**100**	99.2	79.2	**100**	99.9	83.9
nursery	**95.4**	**95.4**	77.8	**94.2**	**94.2**	79.3	**95.2**	**95.2**	79.1
pima	**68.5**	**68.5**	73.4	**68.4**	**68.4**	76.9	67.3	**67.5**	77.0
planning	**48.8**	**48.8**	77.5	**59.3**	**59.3**	86.8	**56.7**	**56.7**	76.4
seeds	93.9	**94.8**	80.3	93.8	**95.2**	88.5	93.8	**93.9**	79.0
shuttle	98.7	**98.8**	80.4	99.0	**99.1**	78.9	97.9	**98.1**	76.2
sonar	**78.0**	**78.0**	80.1	**74.6**	**74.6**	86.7	**80.4**	76.4	78.1
synthetic	**97.2**	**97.2**	78.0	**94.2**	**94.2**	82.8	**96.2**	**96.2**	79.1
tictac	98.6	**98.8**	77.7	98.5	**98.6**	79.0	**98.4**	**98.4**	78.0
vehicle	79.5	**80.7**	76.7	**79.6**	**79.6**	77.3	**80.6**	78.7	76.6
voting	**94.2**	**94.2**	79.8	**86.0**	85.1	81.6	—	—	—
wine	97.8	**97.9**	81.3	**98.6**	**98.6**	89.3	**98.6**	**98.6**	83.7
Avg.	**80.6**	**80.6**	85.2	**80.2**	**80.2**	81.1	**79.9**	79.6	76.2

ACC=81.2% slightly higher than DC-DELM (80.4%). For instance, in `hepatitis` dataset DELM and DC-DELM achieves ACC=76.1% and 81.3%, respectively, while in `connect-4` dataset, DC-ELM outperformed DELM with ACC=77.4% and 76.6%, respectively, with β=81.5%.

Table 4.12: Average accuracy and β of ELM, MA-ELM, EMA-ELM and DC-ELM.

ELM	MA-ELM		EMA-ELM		DC-ELM	
ACC	ACC	β	ACC	β	ACC	β
79.9	79.9	96.2	80.0	80.1	80.0	83.4

Table 4.11 reports the comparison of MELM, RSELM and SELM with and without DC. The DC-MELM (ACC=80.6%) equals MELM in terms of performance leading to a high average β up to 85.2%. The comparison of DC-RSELM and RSELM reports a β value ranging from 76.4% to 89.3%, also with the same average ACC=80.2%. Finally, the average ACC was 79.9% and 79.6% using SELM and DC-SELM, also with a high average β=76.2%.

4.8 Discussion

The results of previous sections report that the three aproaches MA-ELM, EMA-ELM and DC-ELM perform fairly well, achieving accuracy slightly higher than the classical ELM tuned using grid-search and accelerating very much the hyper-parameter tuning and the time spent in training and tuning. Table 4.12 reports that average values of accuracy and β of ELM, MA-ELM, EMA-ELM and DC-ELM compiled from Tables 4.3, 4.6 and 4.9. The average accuracy is very similar using ELM (79.9%) or the proposed methods MA-ELM, EMA-ELM and DC-ELM (79.9% and 80%). This is expectable, because our approaches only reduce the number of H values tried during tuning, but using the same collection of H values as ELM, so the accuracy can be only slightly better. However, the percentage β of reduction in time with respect to ELM is very high in the three methods, although much higher in MA-ELM (96.2%) compared to EMA-ELM and DC-ELM (80% and 83.4%), so the first approach seems to be faster while keeping the same accuracy as the original ELM.

CHAPTER 5

CONCLUSIONS

This dissertation presents three new algorithms based on the extreme learning machine (ELM) that are oriented to solve several defficiencies of these neural networks. The first problem is related to the limitation of the ELM for the classification of large-scale datasets. This limitation is caused by the need of perform generalized inversion of a matrix that scales with the number N of training patterns and with the number H of hidden neurons. With large-scale classification problems, the own matrix can not ne stored in memory due to its large size. Besides, it is even less practical nor factible to perform the generalized inversion of such a large matrix, because the complexity of this problems scales with $\mathscr{O}(N^p)$ with $2 \leq p \leq 3$. The need to perform hyper-parameter tuning over the size H of the hidden layer also adds an important overload and increases the complexity with large-scale datasets, because training and test of the ELM network must be repeated several times in order to select an appropiated size.

In order to solve these drawbacks, the chapter 2 of the current thesis proposes the **quick extreme learning machine** (QELM), a version of ELM designed for large-scale classification problems with many patterns (in our experiments, up to 31 millions) in general purpose computers without any special software (such as pre-trained neural networks or big data platforms) nor hardware (such as graphic processing units, GPUs), something that is not currently available in the literature and therefore exhibits a high practical interest [12]. The classification over large-scale datasets is performed in QELM by replacing the hyper-parameter tuning of the number H of hidden neurons, that usually represents an important overload because it requires to train and test the network several times, by an bounded estimation of H from the number N of training patterns.

On the other hand, instead of performing the generalized inversion of the hidden activation matrix $\mathbf{Y}$, whose size is $H \times N$, the QELM uses an alternative activation matrix of size bounded both on the numbers of rows and columns, avoiding and indefinite growing of the matrix with the dataset size. First, the number H of rows of $\mathbf{Y}$ is required to be $H \leq \lfloor \eta N_0 \rfloor$=2,250 neurons, with η=0.15 and N_0=15,000 training patterns, so the value of H is upper bounded for large N. Second, the number M of columns of $\mathbf{Y}$ is required to be $M \leq N_1 = 5,000$ prototypes, created using an efficient on-line version of the K-means clustering algorithm from the training patterns, instead of the original training patterns as in ELM, so $\mathbf{Y}$ has at most N_1 =5,000 columns when $N \geq N_1$, independently on N. Thus, the maximum size of $\mathbf{Y}$ is $\eta N_0 \times N_1$, or $2,250 \times 5,000$.

Both changes with respect to ELM allow QELM to process datasets with millions of patterns, because only less than 5,000 prototypes are stored, although the time required by the prototype calculation raises with N. Consequently, QELM is able to classify datasets up to 31 millions of patterns (23 millions of training patterns), 30,000 inputs and 131 classes, while the standard ELM can only manage datasets up to 5 million patterns (3,7 millions of training patterns) and even the linear SVM can not execute in three of the largest datasets. The QELM performance, in terms of Cohen kappa (in %) is only 2 and 7 points below ELM and radial SVM, respectively, in small datasets. In large datasets, the kappa of QELM is only 2 points below ELM and 13 points above the linear SVM. Considering speed, QELM is 36 and 154 times faster than ELM and radial SVM, respectively, on the small datasets. In the large datasets, QELM is so fast as the ELM without hyper-parameter tuning (i.e., using H =500), but the latter fails in 40% of the datasets. Compared to the linear SVM, the QELM is 14 times faster. Moreover, the time spent by QELM is relatively stable when the size of the training set increases, due to the limitation on the matrix size. The sensitivity of QELM with respect to its hyper-parameters (e.g. the maximum number N_1 of prototypes) is low, so no tuning is required for a good performance.

The second algorithmic proposal, developed in chapter 3 of this thesis, is relative to the random choice of biases for the neurons of the hidden layer of the ELM. The current work discusses the effectiveness of this random choice and proposes an improved algorithm, named **confidence-random-based extreme learning machine** (CRB-ELM), that makes a proper selection of the random input bias [13]. Once a confidence level F (in %) is defined (e.g., 96%), a confidence interval is determined for the bias of the hidden neurons of the ELM using the mean and standard deviation of the input data (method D), of the input weights (W) or of the product of input data and weights (WD). This confidence interval is created in such a way that F-% of the bias values, that are randomly set, are inside the interval limits. The random initialization of the bias values also follow four different methods to split the data population: the original values (method NCI), the absolute value of the original data (ACI), the upper-lower bounds of the confidence interval (NULCI) and the absolute value of the upper-lower bounds (AULCI). Another alternative method also proposed in this thesis, named **confidence-random-bias singular-value-decomposition extreme learning machine** (CRB-SVD-ELM), uses one of the matrices provided by the singular value decomposition of the true input data as the new input data for the calculation of the confidence interval for the random bias. The results report that the proposed CRB-ELM is a particularly attractive option for solving complex classification and regression problems in the presence of limited computational storage resources. In fact, all the variants of CRB-ELM (WD, D and W) bring significant performance advantages in real applications either with small and large datasets compared to the classical ELM network (about 2% more accuracy in average over the classification tasks, and 0.2 points below in RMSE for the regression tasks) and to the base projection machine (BPVM), that is based on singular value decomposition.

Although the QELM already offers an alternative for hyper-parameter tuning, it is designed mainly for large-scale datasets and brings a small loss in performance compared to the ELM with standard grid-search tuning on small and medium sized datasets. The third algorithmic proposal of the current thesis (chapter 4) uses moving average and exponential moving average statistical tools and the divide-and-conquer strategy in order to make more efficient the search of an optimal value H for the number of hidden neurons. The **moving average extreme learning machine** (MA-ELM) uses MA to estimate the accuracy of ELM trained with different H values, saving an important number of H values and, consequently, of ELM trainings, and thus reducing the time spent in hyper-parameter tuning compared to the classical ELM. As an alternative, the **exponential moving average extreme learning machine** (EMA-ELM) uses EMA instead of MA to perform this estimation with more sensitivity with respect to the changes in the ELM accuracy for each value of H. The **divide-and-conquer extreme learning machine** (DC-ELM) uses this strategy to avoid trying lots of numbers of hidden neurons, and also trainings of the ELM network, by dividing the set of H values in two subsets, evaluating the accuracy in their largest values and selecting the subset with the highest accuracy value. The process is repeated recursively until the current set has only one H value, that is selected for testing. The methods MA, EMA and DC have been also applied to the well-known constrained, constrained sum and constrained difference extreme learning machines (CELM, CSELM and DELM, respectively), and to the mixed, random sum and mixed sum extreme learning machines (MELM, ESELM and SELM, respectively). The results for classification datasets report that the same or even higher accuracy is achieved when the proposed methods are used, compared to the ELM, constrained and mixed ELM. The value selected for H is very often the same as the existing ELM version, but with a very important reduction in the number of H values for that ELM was trained, and in the training-tuning time, that reaches 98% in many cases, i.e., the proposed versions may be until 50 times faster than ELM. Although MA-ELM, EMA-ELM and DC-ELM achieve accuracies very similar to ELM, the MA-ELM is the fastest one, reducing the time an average of 96.2% and being about 26 times faster than ELM. The three methods may bring significant performance advantages in applications where factors limit the number H of hidden neurons and specified biases, such as a multiuser eye-tracking systems [8]. In such cases, and in other analogous real-life applications, the MA-ELM, EMA-ELM, and DC-ELM approaches can deliver improved performance compared to existing alternatives.

The future research includes to extend the capabilities of QELM to datasets with even larger number of inputs, and to integrate the three proposed versions (QELM, CRB-ELM and MA-ELM) on a optimal ELM that allows an efficient and accurate prediction for datasets of any size. As well, we will also research in optimized ways of calculating the generalized inverse for large matrices, in order to remove the main bottleneck of ELM and to extend its use for any machine learning problem.

APPENDIX A

LISTADO DE PUBLICACIÓNS

As publicacións asociadas á tese son as seguintes:

1. A. Albtoush, M. Fernández-Delgado, E. Cernadas, S. Barro. Quick extreme learning machine for large-scale classification. Revista *Neural Computing and Applications*, Springer, pp. 1-16. Enero 2022. ISSN 1433-3058. DOI: https://doi.org/10.1007/s00521-021-06727-8. Índice de impacto 5.606 (2020), posición 31 de 139 en Computer Science, Artificial Intelligence (Q1). Os contidos desta publicación descríbense no capítulo 2 da tese.

2. A. Albtoush, M. Fernández-Delgado, E. Cernadas, S. Barro. "Extreme learning machine with confidence interval based bias initialization". II IEEE International Conference on Intelligent Data Science Technologies and Applications (IDSTA), 2021, Tartu (Estonia), 15-16 noviembre 2021, pp 1–8. DOI https://doi.org/10.1109/IDSTA53674.2021.9660822. Os contidos desta publicación descríbense no capítulo 3 da tese. Este artigo recibiu o **Best Paper Award** neste congreso.

Os co-autores das publicacións Eva Cernadas e Senén Barro pertencen ao Departamento de Electrónica e Computación, Universidade de Santiago de Compostela. Estas publicacións foron realizadas exclusivamente polo doutorando, polo tanto a súa contribución é do 100% das mesmas, é orixinal e propia do doutorando.

Bibliography

[1] M. Abd Shehab and N. Kahraman. A weighted voting ensemble of efficient regularized extreme learning machine. *Comput Elec Engin*, 85:106639, 2020.

[2] A.L. Afzal, N.K. Nair, and S. Asharaf. Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection. *Patt Anal Appl*, 24:11–19, 2021.

[3] I. Ahmad, M. Basheri, M.J. Iqbal, and A. Rahim. Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection. *IEEE Access*, 6:33789–33795, 2018.

[4] Z. Akram-Ali-Hammouri, M.Fernández-Delgado, E. Cernadas, and S. Barro. Fast support vector classification for large-scale problems. *IEEE T Pattern Anal*, 2021.

[5] A. Akusok, K.M. Björk, Y. Miche, and A. Lendasse. High-performance extreme learning machines: a complete toolbox for big data applications. *IEEE Access*, 3:1011–1025, 2015.

[6] A. Akusok, Y. Miche, K. Björk, R. Nian, P. Lauren, and A. Lendasse. Evaluating confidence intervals for ELM predictions. In *Proc. ELM-2015*, volume 2, pages 413–422. Springer, 2016.

[7] A. Akusok, Y. Miche, K.M. Björk, and A. Lendasse. Per-sample prediction intervals for extreme learning machines. *Intl J Mach Learn Cyb*, 10(5):991–1001, 2019.

[8] A. Al-Btoush, M. Abbadi, A. Hassanat, A. Tarawneh, A. Hasanat, and V. Prasath. New features for eye-tracking systems: Preliminary results. In *Intl Conf Inf Comm Syst*, pages 179–184. IEEE, 2019.

[9] P. Alaba, S. Popoola, L. Olatomiwa, M. Akanle, O. Ohunakin, E. Adetiba, O Alex, A. Atayero, and W. Daud. Towards a more efficient and cost-sensitive extreme learning machine: A state-of-the-art review of recent trend. *Neurocomputing*, 350:70–90, 2019.

[10] O. Alade, A. Selamat, and R. Sallehuddin. A review of advances in extreme learning machine techniques and its applications. In *Intl Conf Reliable Inf Comm Technol*, pages 885–895. Springer, 2017.

[11] O. Alade, A. Selamat, and R. Sallehuddin. A review of advances in extreme learning machine techniques and its applications. In *Intl Conf Reliable Inf and Comm Technol*, pages 885–895, 05 2018.

[12] A. Albtoush, M. Fernández-Delgado, E. Cernadas, and S. Barro. Quick extreme learning machine for large-scale classification. *Neural Comput Appl*, pages 1–16, 2022.

[13] A. Albtoush, M. Fernández-Delgado, E. Cernadas, and S. Barro. Extreme learning machine with confidence interval based bias initialization. In *IEEE Intl Conf Intel Data Sci Technol Appl (IDSTA)*, pages 1–8. IEEE, 11 2021.

[14] F. Aldakheel, R. Satari, and P. Wriggers. Feed-forward neural networks for failure mechanics problems. *Appl Sci*, 11(14):6483, 2021.

[15] P. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE T Inf Theory*, 44(2):525–536, 1998.

[16] J. Cao and Z. Lin. Extreme learning machines on high dimensional and large data applications: a survey. *Math Probl Engin*, 2015:1–13, 2013.

[17] J. Cao, Z. Lin, G.B. Huang, and N. Liu. Voting based extreme learning machine. *Inf Sci*, 185(1):66–77, 2012.

[18] L. Cao, Y. Yue, Y. Zhang, and Y. Cai. Improved crow search algorithm optimized extreme learning machine based on classification algorithm and application. *IEEE Access*, 9:20051–20066, 2021.

[19] J. Carletta. Assessing agreement on classification tasks: the kappa statistic. *Comput Lingüist*, 22(2):249–254, 1996.

[20] C.C. Chang and C.J. Lin. LIBSVM: A library for support vector machines. *ACM T Intel Systs Technol*, 2:27:1–27:27, 2011.

[21] H. Chen, Q. Zhang, J. Luo, Y. Xu, and X. Zhang. An enhanced bacterial foraging optimization and its application for training kernel extreme learning machine. *Appl Soft Comput*, 86:105884, 2020.

[22] T. Colton. *Statistics in medicine*. Little Brown and Co, 1974.

[23] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proc ACM Sympos Theory Comput*, pages 1–6, 1987.

[24] P. Courrieu. Fast computation of Moore-Penrose inverse matrices. *Neur Inf Proc Lett Rev*, 8:25–29, 2005.

[25] D. Cui, G.B. Huang, and T. Liu. ELM based smile detection using distance vector. *Patt. Recogn.*, 79:356–369, 2018.

[26] D. Cui, G. Zhang, W. Han, L. Lekamalage Chamara Kasun, K. Hu, and G.B. Huang. Compact feature representation for image classification using elms. In *Intl Conf Comput Vision Workshops*, pages 1015–1022, 2017.

[27] P. de Campos Souza, L. Bambirra Torres, G. Lacerda Silva, A. Braga, and E. Lughofer. An advanced pruning method in the architecture of extreme learning machines using L1-regularization and bootstrapping. *Electronics*, 9(5):811, 2020.

[28] W.Y. Deng, Z. Bai, G.B. Huang, and Q.H. Zheng. A fast SVD-hidden-nodes based extreme learning machine for large-scale data analytics. *Neural Netw*, 77:14–28, 2016.

[29] W.Y. Deng, Q.H. Zheng, and Z.M. Wang. Projection vector machine. *Neurocomputing*, 120:490–498, 2013.

[30] M.L.D. Dias, L.S. de Sousa, A.R. Neto, and A.L. Freire. Fixed-size extreme learning machines through simulated annealing. *Neural Proc Lett*, 48(1):135–151, 2018.

[31] S. Ding, X. Xu, and R. Nie. Extreme learning machine and its applications. *Neural Comput Appl*, 25(3-4):549–556, 2014.

[32] J. Duan, Y. Ou, J. Hu, Z. Wang, S. Jin, and C. Xu. Fast and stable learning of dynamical systems based on extreme learning machine. *IEEE T Syst Man Cybern A: Syst*, 49(6):1175–1185, 2017.

[33] M. Duan, K. Li, X. Liao, and K. Li. A parallel multiclassification algorithm for big data using an extreme learning machine. *IEEE T Neur Net Lear*, 29(6):2337–2351, 2018.

[34] A.H. Elsheikh, A.I. Sabab, M. Abd Elaziz, S. Lud, S. Shanmugan, T. Muthuramalingam, R. Kumar, A.O. Mosleh, F.A. Essa, and T.A. Shehabeldeen. Deep learning-based forecasting model for COVID-19 outbreak in Saudi Arabia. *Proc Saf Environ*, 149:223–233, 2021.

[35] Ömer Faruk Ertuğrul and Mehmet Emin Tağluk. A fast feature selection approach based on extreme learning machine and coefficient of variation. *Turkish J Elec Eng Comput Sci*, 25(4):3409–3420, 2017.

[36] R.E. Fan, K.W. Chang, C.J. Hsieh, X.R. Wang, and C.J. Lin. LIBLINEAR: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, 2008.

[37] G. Feng, G.B. Huang, Q. Lin, and R. Gay. Error minimized extreme learning machine with growth of hidden nodes and incremental learning. *IEEE T Neural Netw*, 20(8):1352–1357, 2009.

[38] Z. Geng, S. Zhao, G.n Tao, and Y. Han. Early warning modeling and analysis based on analytic hierarchy process integrated extreme learning machine (AHP-ELM): Application to food safety. *Food Control*, 78:33–42, 2017.

[39] M. Guo, Y. Ma, X. Yang, and R. Mankin. Detection of damaged wheat kernels using an impact acoustic signal processing technique based on gaussian modelling and an improved extreme learning machine algorithm. *Biosyst Engin*, 184:37–44, 2019.

[40] A. Hashmi and T. Ahmad. GP-ELM-RNN: Garson-pruned extreme learning machine based replicator neural network for anomaly detection. *J King Saud Univ-Comput Inf Sci*, 2019.

[41] S. Hassan, M.A. Khanesar, J. Jaafar, and A. Khosravi. Optimal parameters of an ELM-based interval type 2 fuzzy logic system: a hybrid learning algorithm. *Neural Comput Appl*, 29(4):1001–1014, 2018.

[42] Q. He, T. Shang, F. Zhuang, and Z. Shi. Parallel extreme learning machine for regression based on MapReduce. *Neurocomputing*, 102:52–58, 2013.

[43] Y. He, Y Tian, Y. Xu, and Q. Zhu. Novel soft sensor development using echo state network integrated with singular value decomposition: Application to complex chemical processes. *Chemometr Intel Lab*, page 103981, 2020.

[44] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Netw*, 2(5):359–366, 1989.

[45] C. Hsieh, S. Si, and I. Dhillon. A divide-and-conquer solver for kernel support vector machines. In *Intl Conf Mach Learn*, pages 566–574. PMLR, 2014.

[46] H.H. Hsu, C.Y. Chang, and C.H. Hsu. Chapter 6 - extreme learning machine and its applications in big data processing. In *Big data analytics for sensor-network collected intelligence*, Intelligent Data-Centric Systems, pages 117–150. Academic Press, 2017.

[47] F. Huang, K. Yin, J. Huang, L. Gui, and P. Wang. Landslide susceptibility mapping based on self-organizing-map network and extreme learning machine. *Engin Geol*, 223:11–22, 2017.

[48] G. Huang, G.B. Huang, S. Song, and K. You. Trends in extreme learning machines: A review. *Neural Netw*, 61:32–48, 2015.

[49] G. Huang, T. Liu, Y. Yang, Z. Lin, S. Song, and C. Wu. Discriminative clustering via extreme learning machine. *Neural Netw*, 70:1–8, 2015.

[50] G.-B. Huang, X. Ding, and H. Zhou. Optimization method based extreme learning machine for classification. *Neurocomputing*, 74(1-3):155–163, 2010.

[51] G.B. Huang. An insight into extreme learning machines: random neurons, random features and kernels. *Cogn Comput*, 6(3):376–390, 2014.

[52] G.B. Huang and L. Chen. Convex incremental extreme learning machine. *Neurocomputing*, 70(16-18):3056–3062, 2007.

[53] G.B. Huang and L. Chen. Enhanced random search based incremental extreme learning machine. *Neurocomputing*, 71(16-18):3460–3468, 2008.

[54] G.B. Huang, L. Chen, C. Siew, et al. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE T Neural Netw*, 17(4):879–892, 2006.

[55] G.B. Huang, Q. Zhu, and C. Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *Intl J Conf Neural Netw*, volume 2, pages 985–990. IEEE, 2004.

[56] G.B. Huang, Q. Zhu, and C. Siew. Real-time learning capability of neural networks. *IEEE T Neural Netw*, 17(4):863–878, 2006.

[57] G.B. Huang, Q.Y. Zhu, and C.K. Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006.

[58] Guang-Bin Huang, Hongming Zhou, Xiaojian Ding, and Rui Zhang. Extreme learning machine for regression and multiclass classification. *IEEE T Syst Man Cyb B*, 42(2):513–529, 2011.

[59] Y. Huang and D. Lai. Hidden node optimization for extreme learning machine. *AASRI Procedia*, 3:375–380, 2012.

[60] A. Iosifidis, A. Tefas, and I. Pitas. Regularized extreme learning machine for large-scale media content analysis. *Procedia Comput Sci*, 53:420–427, 2015.

[61] A. Iosifidis, A. Tefas, and I. Pitas. Approximate kernel extreme learning machine for large scale data classification. *Neurocomputing*, 219:210–220, 2017.

[62] K. Khan, E. Ratner, R. Ludwig, and A. Lendasse. Feature bagging and extreme learning machines: machine learning with severe memory constraints. In *Intl J Conf Neur Netw*, pages 1–7, 2020.

[63] B.E. Köktürk-Güzel and S. Beyhan. Symbolic regression based extreme learning machine models for system identification. *Neural Proc Lett*, 53:1565–1578, 2021.

[64] Y. Kongsorot, P. Horata, and P. Musikawan. An incremental kernel extreme learning machine for multi-label learning with emerging new labels. *IEEE Access*, 8:46055–46070, 2020.

[65] J. Lai, X. Wang, R. Li, Y. Song, and L. Lei. BD-ELM: A regularized extreme learning machine using biased dropconnect and biased dropout. *Math Probl Engin*, 2020, 2020.

[66] Y. Lan, Y. Soh, and G.B. Huang. A constructive enhancement for online sequential extreme learning machine. In *Intl J Conf Neural Netw*, pages 1708–1713. Ieee, 2009.

[67] Y. Lan, Y. Soh, and G.B. Huang. Constructive hidden nodes selection of extreme learning machine for regression. *Neurocomputing*, 73(16-18):3191–3199, 2010.

[68] Y. Lan, Y.C. Soh, and G.B. Huang. Ensemble of online sequential extreme learning machine. *Neurocomputing*, 72(13-15):3391–3395, 2009.

[69] M. Larrea, A. Porto, E. Irigoyen, A.J. Barragán, and J.M. Andújar. Extreme learning machine ensemble model for time series forecasting boosted by PSO: application to an electric consumption problem. *Neurocomputing*, 72(13-15):3391–3395, 2009.

[70] J. Li, X. Shi, Z. You, H. Yi, Z. Chen, Q. Lin, and M. Fang. Using weighted extreme learning machine combined with scale-invariant feature transform to predict protein-protein interactions from protein evolutionary information. *IEEE/ACM T Comput Biol Bionf*, 17(5):1546–1554, 2020.

[71] L. Li, G. Wang, G. Wu, and Q. Zhang. An experimental evaluation of extreme learning machines on several hardware devices. *Neural Comput Appl*, 32:14385–14397, 2020.

[72] Y. Li, Y. Zeng, Y. Qing, and G.B. Huang. Learning local discriminative representations via extreme learning machine for machine fault diagnosis. *Neurocomputing*, 409:275–285, 2020.

[73] Z. Li, L. Wei, W. Li, L. Wei, M. Chen, M. Lv, X. Zhi, C. Wang, and N. Gao. Research on DDoS attack detection based on ELM in IoT environment. In *Intl Conf Softw Engin and Service Sci*, pages 144–148. IEEE, 2019.

[74] N. Liang, P. Saratchandran, G.B. Huang, and N. Sundararajan. Classification of mental tasks from EEG signals using extreme learning machine. *Intl J Neural Syst*, pages 29–38, 2006.

[75] N.Y. Liang, G.B. Huang, P. Saratchandran, and N. Sundararajan. A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE T Neural Netw*, 17(6):1411–1423, 2006.

[76] B. Liu, S. Yan, H. You, Y. Dong, J. Li, Y. Li, J. Lang, and R. Gu. An ensembled RBF extreme learning machine to forecast road surface temperature. In *Intel Conf Mach Learn Appl (ICMLA)*, pages 977–980. IEEE, 2017.

[77] P. Liu, Y. Huang, L. Meng, S. Gong, and G. Zhang. Two-stage extreme learning machine for high-dimensional data. *Intl J Mach Learn Cyb*, 7(5):765–772, 2016.

[78] X. Liu, C. Gao, and P. Li. A comparative analysis of support vector machines and extreme learning machines. *Neural Netw*, 33:58–66, 2012.

[79] D. Lowe. Adaptive radial basis function nonlinearities, and the problem of generalisation. In *IEE Intl Conf Artif Neural Netw (No. 313)*, pages 171–175. IET, 1989.

[80] C.J. Lu and L.J. Kao. A clustering-based sales forecasting scheme by using extreme learning machine and ensembling linkage methods with applications to computer server. *Eng. Appl. Artif. Intel.*, 55:231–238, 2016.

[81] S. Lu, G. Zhang, and X. Wang. A rank reduced matrix method in extreme learning machine. In J. Wang, G.G. Yen, and M. Polycarpou, editors, *Advances in Neural Networks-ISNN 2012*, pages 72–79, 2012.

[82] J.M. López-Guede, A. Izquierdo, J. Estévez, and M. Graña. Active learning for road lane landmark inventory with V-ELM in highly uncontrolled image capture conditions. *Neurocomputing*, 438:259–269, 2021.

[83] M. Mahmoudi and S. Baroumand. Modeling the stochastic mechanism of sensor using a hybrid method based on seasonal autoregressive integrated moving average time series and generalized estimating equations. *ISA T*, 2021.

[84] Z. Mai, Y. Chen, and L. Du. A novel blind mmwave channel estimation algorithm based on ML-ELM. *IEEE Comm Lett*, 25(5):1549–1553, 2021.

[85] W. Mao, Y. Zheng, X. Mu, and J. Zhao. Uncertainty evaluation and model selection of extreme learning machine based on Riemannian metric. *Neural Comput Appl*, 24:1613–1625, 2014.

[86] N. Marques and C. Gomes. Implementing an intelligent moving average with a neural network. In *European Conf Artif Intel*, pages 1129–1130. IOS Press, 2010.

[87] C. Merkel and D. Kudithipudi. Neuromemristive extreme learning machines for pattern classification. In *IEEE Comput Soc Ann Sympos on VLSI*, pages 77–82. IEEE, 2014.

[88] Y. Miche, A. Sorjamaa, and A. Lendasse. OP-ELM: theory, experiments and a toolbox. In *Int Conf Artif Neural Netw*, pages 145–154. Springer, 2008.

[89] W. Niu, Z. Feng, B. Feng, Y. Min, C. Cheng, and J. Zhou. Comparison of multiple linear regression, artificial neural network, extreme learning machine, and support vector machine in deriving operation rule of hydropower reservoir. *Water*, 11(1):88, 2019.

[90] L. Oneto, F. Bisio, E. Cambria, and D. Anguita. Statistical learning theory and ELM for big social data analysis. *IEEE Comput Intel Mag*, 11(3):45–55, 2016.

[91] T. Ouyang. Feature learning for stacked ELM via low-rank matrix factorization. *Neurocomputing*, 448:82–93, 2021.

[92] C. Perales-González, M. Carbonero-Ruz, J. Pérez-Rodríguez, D. Becerra-Alonso, and F. Fernández-Navarro. Negative correlation learning in the extreme learning machine framework. *Neural Comput Appl*, 32(17):13805–13823, 2020.

[93] Y. Qing, Y. Zeng, Y. Li, and G.B. Huang. Deep and wide feature based extreme learning machine for image classification. *Neurocomputing*, 412:426–436, 2020.

[94] B. Qu, B. Lang, J. Liang, A. Qin, and O. Crisalle. Two-hidden-layer extreme learning machine for regression and classification. *Neurocomputing*, 175:826–834, 2016.

[95] S. Rajpal, N. Kumar, and A. Rajpal. COV-ELM classifier: An extreme learning machine based identification of COVID-19 using chest-ray images. *arXiv:2007.08637*, 2020.

[96] Y. Rong, H.and Ong, A. Tan, and Z. Zhu. A fast pruned-extreme learning machine for classification problem. *Neurocomputing*, 72(1-3):359–366, 2008.

[97] X. Rongjun, I. Khalil, S. Badsha, and M. Atiquzzaman. Collaborative extreme learning machine with a confidence interval for P2P learning in healthcare. *Comput Netw*, 149:127–143, 2019.

[98] F. Rosenblatt. *Perceptions and the theory of brain mechanisms*. Spartan books, 1962.

[99] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[100] D. Serre. Elementary theory. *Matrices: Theory and Applications*, pages 1–14, 2002.

[101] Z. Shang and J. He. Confidence-weighted extreme learning machine for regression problems. *Neurocomputing*, 148:544–550, 2015.

[102] Q. She, B. Hu, H. Gan, Y. Fan, T. Nguyen, T. Potter, and Y. Zhang. Safe semi-supervised extreme learning machine for EEG signal classification. *IEEE Access*, 6:49399–49407, 2018.

[103] K. Sheela and S. Deepa. Review on methods to fix number of hidden neurons in neural networks. *Math Probl Engin*, 2013, 2013.

[104] S. Shoumo, M. Dhruba, S. Hossain, N. Ghani, H. Arif, and S. Islam. Application of machine learning in credit risk assessment: a prelude to smart banking. In *IEEE Region 10 Conf*, pages 2023–2028. IEEE, 2019.

[105] T. Similä and J. Tikka. Multiresponse sparse regression with application to multidimensional scaling. In *Int Conf Artif Neural Netw*, pages 97–102. Springer, 2005.

[106] A. Šimundić. Confidence interval. *Biochem. medica*, 18(2):154–161, 2008.

[107] S. Song, M. Wang, and Y. Lin. An improved algorithm for incremental extreme learning machine. *Syst Sci Control Engin*, 8(1):308–317, 2020.

[108] E. Soria-Olivas, J. Gomez-Sanchis, J. Martin, J. Vila-Frances, M. Martinez, J. Magdalena, and A. Serrano. BELM: Bayesian extreme learning machine. *IEEE T Neural Netw*, 22(3):505–509, 2011.

[109] V. Strassen. Gaussian elimination is not optimal. *Numer Math*, 13:354–356, 1969.

[110] N. Surantha, T.F. Lesmana, and S.M. Isa. Sleep stage classification using extreme learning machine and particle swarm optimization for healthcare big data. *J Big Data*, 8(1):1–17, 2021.

[111] N. Tak. Meta fuzzy functions based feed-forward neural networks with a single hidden layer for forecasting. *J Stat Comput Simul*, pages 1–17, 2021.

[112] M. Van Heeswijk, Y. Miche, T. Lindh-Knuutila, P. Hilbers, T. Honkela, E. Oja, and A. Lendasse. Adaptive ensemble models of extreme learning machines for time series prediction. In *Intl Conf Artif Neural Netw*, pages 305–314. Springer, 2009.

[113] Y. Wan, S. Song, G. Huang, and S. Li. Twin extreme learning machines for pattern classification. *Neurocomputing*, 260:235–244, 2017.

[114] H. Wang, Q. He, T. Shang, F. Zhuang, and Z. Shi. Extreme learning machine ensemble classifier for large-scale data. In *Proc. ELM-2014*, volume 1, pages 151–161. Springer, 2015.

[115] R. Wang, C. Chow, Y. Lyu, V. Lee, S. Kwong, Y. Li, and J. Zeng. TaxiRec: Recommending road clusters to taxi drivers using ranking-based extreme learning machines. *IEEE T Knowl Data Engin*, 30(3):585–598, 2017.

[116] S. Wang, E. Zhu, J. Yin, and F. Porikli. Video anomaly detection and localization by local motion based joint video representation and OCELM. *Neurocomputing*, 277:161–175, 2018.

[117] Y. Wang, F. Cao, and Y. Yuan. A study on effectiveness of extreme learning machine. *Neurocomputing*, 74:2483–2490, 2011.

[118] J. Xin, Z. Wang, L. Qu, and G. Wang. Elastic extreme learning machine for big data classification. *Neurocomputing*, 149:464–471, 2015.

[119] Y. Xing, X. Ban, X. Liu, and Q. Shen. Large-scale traffic congestion prediction based on the symmetric extreme learning machine cluster fast learning method. *Symmetry*, 11(6(730)):1–19, 2019.

[120] Z. Xu, M. Yao, Z. Wu, and W. Dai. Incremental regularized extreme learning machine and it's enhancement. *Neurocomputing*, 174:134–142, 2016.

[121] H. Yıldırım and M. Özkale. An enhanced extreme learning machine based on Liu regression. *Neural Proc. Let.*, pages 1–22, 2020.

[122] D. Yu and L. Deng. Efficient and effective algorithms for training single-hidden-layer neural networks. *Patt Recogn Lett*, 33(5):554–558, 2012.

[123] J. Zhai, Q. Shao, and X. Wang. Architecture selection of ELM networks based on sensitivity of hidden nodes. *Neural Proc Lett*, 44(2):471–489, 2016.

[124] J. Zhai, J. Wang, and X. Wang. Ensemble online sequential extreme learning machine for large data set classification. In *IEEE Intl Conf Syst Man Cyb*, pages 2250–2255, 2014.

[125] J. Zhai, S. Zhang, and C. Wang. The classification of imbalanced large data sets based on MapReduce and ensemble of ELM classifiers. *Intl J Mach Learn Cybern*, 8:1009–1017, 2017.

[126] Y. Zhai, Y. Ong, and I. Tsang. The emerging "big dimensionality". *IEEE Comput Intel Mag*, 9(3):14–26, 2014.

[127] J. Zhang, L. Feng, and B. Wu. Local extreme learning machine: local classification model for shape feature extraction. *Neural Comput Appl*, 27(7):2095–2105, 2016.

[128] L. Zhang, X. W, G.B. Huang, T. Liu, and X. Tan. Taste recognition in e-tongue using local discriminant preservation projection. *IEEE T Cyb*, 49(3):947–960, 2018.

[129] D. Zheng, Z. Hong, N. Wang, and P. Chen. An improved LDA-based ELM classification for intrusion detection algorithm in IoT application. *Sensors*, 20(6):1706, 2020.

[130] Z. Zhou, C. Wang, Z. Zhu, Y. Wang, and D. Yang. Sliding mode control based on a hybrid grey-wolf-optimized extreme learning machine for robot manipulators. *Optik*, 185:364–380, 2019.

[131] Q. Zhu, A. Qin, P. Suganthan, and G.B. Huang. Evolutionary extreme learning machine. *Patt Recogn*, 38(10):1759–1763, 2005.

[132] S. Zhu, H. Wang, H. Lv, and H. Zhang. Augmented online sequential quaternion extreme learning machine. *Neural Proc Lett*, 53:1161–1186, 2021.

[133] W. Zhu, J. Miao, and L. Qing. Constrained extreme learning machine: A novel highly discriminative random feedforward neural network. In *Intl J Conf Neural Netw*, pages 800–807, 2014.

[134] W. Zhu, J. Miao, and L. Qing. Constrained extreme learning machines: A study on classification cases. *arXiv:1501.06115*, 2015.

[135] W. Zong, G.B. Huang, and Y. Chen. Weighted extreme learning machine for imbalance learning. *Neurocomputing*, 101:229–242, 2013.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

CAMPUS VIDA CAMPUS DE EXCELENCIA INTERNACIONAL

The extreme learning machine (ELM) is a popular neural network that has two major drawbacks for large-scale data sets: the need of tuning of the number of hidden neurons, and the pseudo-inversion of the hidden activation matrix. This thesis proposes algorithms that keep the simplicity and speed of the ELM network and: 1) avoid tuning and bound the size of hidden activation matrix; 2) raise the ELM performance with a suitable choice of the random bias values; 3) speeds up the hyper-parameter tuning by reducing the number of training executions required.