Modelo Neuronal de Estimación para el Esfuerzo de Desarrollo en Proyectos de Software (MONEPS)

Mario G. Almache C., Jenny A. Ruiz R., Geovanny Raura y Efraín R. Fonseca C.

Universidad de las Fuerzas Armadas ESPE, Sangolquí, Ecuador
{mgalmache, jaruiz, jgraura, erfonseca} @espe.edu.ec

Resumen. La estimación temprana del esfuerzo para la construcción de un producto software, es crucial en la previsión del costo y tiempo necesarios para el desarrollo de software. Los modelos y técnicas para la estimación del esfuerzo presentan, como principal inconveniente, la poca precisión en las predicciones realizadas, y generalmente se hace una mínima consideración de los aspectos no funcionales del software. Se propone la construcción de un modelo de estimación para el esfuerzo en el desarrollo de software, denominado MONEPS, que pretende mejorar la precisión en la estimación del esfuerzo, utilizando una Red Neuronal Artificial (RNA) en Backpropagation, cuya capa de entrada se estructura sobre la base de un conjunto de características y atributos tomados de la norma ISO 25000 de la calidad del software. La RNA fue entrenada con datos recopilados de aplicaciones desarrolladas en el ámbito académico, de las cuales se conocían sus tiempos de desarrollo y costos asociados. Las estimaciones de tiempo y costo, para dos casos de prueba, muestran más precisión en el modelo neuronal, en comparación con los modelos Cocomo-81 y Cocomo-II. MONEPS ha logrado la convergencia de aspectos funcionales y no funcionales para mejorar la precisión en la estimación de dicho esfuerzo.

Palabras Clave: Software, Inteligencia Artificial, Redes Neuronales, Estimación del Esfuerzo.

1 Introducción

La estimación del esfuerzo es un aspecto de particular importancia en la elaboración de los proyectos de software [1]. Las pequeñas y medianas empresas de software se enfrentan al desafío de seleccionar, de una parte, el método de estimación más adecuado para su contexto, y de otra parte, adoptar mejores prácticas que permitan ir consolidando una base histórica de estimación, con el propósito de disminuir cada vez más la brecha entre los valores estimados y los valores reales [2].

La estimación del esfuerzo para el desarrollo de software es una actividad compleja y de poca precisión en los resultados obtenidos [3]. Sin embargo, se han desarrollado varias herramientas comerciales para la estimación de costos del software, tales como [4]: COCOMO II [5], CoStar [6], CostModeler [7], CostXpert [8], KnowledgePlan®

[9], SEER [10], y SoftCost-R [11]. Así mismo, existen algunas herramientas de estimación de costos que ya no son utilizadas o están en declive [12], debido a su temprana aparición; así tenemos: CheckPoint [13], ESTIMACS [14], REVIC [15] y SPQR/20 [16]. La presencia de estas herramientas, al parecer no ha solucionado del todo la problemática de la estimación, al contario ésta se ha potenciado debido al poco mantenimiento de las herramientas y modelos disponibles [17].

En el presente artículo se presenta una propuesta de Modelo Neuronal de Estimación para el Esfuerzo de Desarrollo en Proyectos de Software (MONEPS), utilizando una RNA en Backpropagation[1], cuya capa de entrada se basa en un conjunto de atributos de la norma de calidad del software ISO 25000 [18]. Esta RNA fue entrenada con datos referidos a aplicaciones desarrolladas en el ámbito académico, de las que se conocían el tiempo empleado para su desarrollo y los costos asociados. La evaluación de MONEPS se llevó a cabo sobre dos aplicaciones específicas, obteniéndose mejoras significativas en las predicciones de tiempo y costo, en relación a los modelos Cocomo-81 [19] y Cocomo-II [20]. Es decir, se logró la convergencia de aspectos funcionales y no funcionales del software para mejorar la precisión en la estimación del esfuerzo en este tipo de aplicaciones.

El resto del documento está organizado de la siguiente manera: En la Sección 2 se pone de manifiesto la importancia de mejorar la precisión en las estimaciones realizadas. También se describen las principales categorías de los modelos de estimación para el esfuerzo en productos de software, enfatizando en las técnicas de inteligencia artificial y en el modelo Cocomo-II. La Sección 3 presenta la propuesta neuronal para la estimación del esfuerzo en el desarrollo de proyectos de software, y se muestran los resultados obtenidos en la fase de entrenamiento de la RNA. En la Sección 4 se muestra el desempeño de MONEPS cuando sus resultados son comparados con aquellos calculados a través de los modelos Cocomo-81 y Cocomo-II. Finalmente, en la Sección 5 se mencionan las principales conclusiones de esta investigación y el trabajo futuro.

2 Antecedentes

2.1 Necesidad de mejorar la precisión en la estimación del esfuerzo

Hasta la fecha no hay un completo entendimiento sobre las relaciones causales entre factores ni de su influencia en el resultado final de la estimación de esfuerzo en software [21]. Los usuarios no están totalmente satisfechos con los resultados obtenidos con los métodos de estimación; los errores en la estimación de esfuerzo son demasiado grandes, lo que lleva a reflexionar sobre si las estimaciones del esfuerzo en software son adecuadas a las necesidades de los líderes de proyectos o dueños de las aplicaciones [22]. Por otro lado, el principal inconveniente de los modelos predictivos basados en líneas de código (p.e. del tipo Cocomo), es que, las líneas de código no pueden medirse hasta que el sistema esté completo [23]. Además, los modelos

[1] Backpropagation es el algoritmo que permite entrenar RNA's con topología conexionista unidireccional.

genéricos como Cocomo fallan a la hora de realizar ajustes de la predicción sin calibración. La necesidad calibración ha sido confirmada en estudios de Kemerer [24], Kitchenham y Taylor [25], Low y Jeffery [26]. La poca atención de los aspectos no funcionales del software ha sido otra característica de los modelos predictivos para estimar el esfuerzo [27].

El clustering [28] es otra técnica de inteligencia artificial utilizada en la estimación del software, para salvar el inconveniente de los modelos paramétricos basados en la utilización de una única ecuación que representa a toda una base de datos de proyectos. Chiu y Huang [29] proponen mejorar la estimación del esfuerzo basado en analogía [30], con la ayuda de algoritmos genéticos y medidas de distancias (p.e. Euclidea, Manhattan, y Minkowski) entre parejas de proyectos. Kumar [31] utilizó una RNA para predecir el esfuerzo en software, con la ayuda de funciones de transferencia de Morlet y Gaussiana. Pichai [32] hace una propuesta neuronal para predecir el esfuerzo, basado en la reducción de características del modelo Cocomo 81. En el 2011, García [33] hace una propuesta neuronal para estimar el tiempo de duración en proyectos de software, recurriendo a un subconjunto de datos obtenido de la organización International Software Benchmarking Standards Group (ISBSG); en dicha propuesta se filtró el conjunto de características disponibles, excluyendo aquellas referentes al cálculo de puntos de función.

2.2 Modelos y técnicas de estimación para el esfuerzo en software

Las técnicas de estimación, pueden ser clasificadas bajo tres principales categorías [34]:

i) *Juicio del experto*: un estimador de proyectos de software usa su experticia basada en información histórica y en proyectos similares para estimar. El principal inconveniente de esta técnica es la dificultad de estandarización en los criterios de los expertos [35] [36].

ii) *Modelos algorítmicos*: es la categoría más popular en la literatura; esos modelos incluyen Cocomo [37], SLIM [38] y SEER-SEM [39]. El factor principal de costo de estos modelos es el tamaño del software, que usualmente está dado en líneas de código. Estos modelos usan fórmulas de regresión lineal o también fórmulas de regresión no lineal. La desventaja de estos modelos radica en la necesidad de hacer ajustes a las predicciones cuando los modelos no están calibrados [40].

iii) *Aprendizaje de máquina*: actualmente, estas técnicas están siendo utilizadas en conjunción o como alternativas a los modelos algorítmicos. Estas técnicas pueden incluir: lógica difusa [41], redes neuronales artificiales [42], minería de datos, sistemas neuro-difusos [43], algoritmos genéticos [44].

La tabla 2.1 muestra un resumen de las principales ventajas e inconvenientes en los enfoques i) y ii) para la estimación del esfuerzo de software.

Tabla 2.1 Principales ventajas y desventajas en dos enfoques para la estimación del esfuerzo de software

Enfoque	Ventajas	Inconvenientes	Aplicación idónea
Modelos algorítmicos	- Entradas y parámetros concretos. - Objetividad. - Eficiencia en cálculos.	- No prestan atención a circunstancias excepcionales. - Rechazan opiniones subjetivas.	Proyectos con escasas alteraciones accidentales, con equipos de desarrollo estables y productos sencillos.
Juicio del experto	- Gran cantidad de opiniones subjetivas. - Consideración de circunstancias excepcionales.	- Dependencia de los expertos. - Posturas de expertos difíciles de adoptar.	Primeras fases de desarrollo del producto.

Varios modelos de estimación que se han publicado desde los años 60 han quedado obsoletos [45], tales como el modelo Aaron (1969), el modelo de Wolverton (1974), el modelo de Walston-Feliz (1977), el modelo Doty (1977); el modelo Putnam (1978) y otros como el modelo SLIM (1979). El modelo COCOMO 81 (1981) y su actualización COCOMO II (1997) han ido evolucionando, siendo la base de las herramientas de estimación existentes en la actualidad [46].

En el 2012, P.K. Suri y Pallavi Ranjan [47] hacen un análisis de los métodos de estimación desde la década de los 70s, concluyendo que en los últimos 5 años se introdujeron diversos métodos en aras de incrementar la precisión de los resultados, tras la identificación de algunos problemas teóricos (Kitchenham, 2009). Para ello, la tendencia es combinar diferentes métodos de estimación, y utilizar a su vez, técnicas de inteligencia artificial, entre las que destacan: lógica difusa, sistemas basados en conocimiento y redes neuronales artificiales [48].

Se cree conveniente aportar con una propuesta neuronal para estimar el esfuerzo de desarrollo en proyectos de software, basada en el uso de la norma ISO 25000 para la calidad de software. MONEPS no tiene una fórmula explícita con coeficientes preestablecidos para el cálculo del esfuerzo; tampoco requiere un número de líneas de código ni puntos de función (como es el caso del modelo Boehm y el modelo Albretch) en el proceso de estimación del esfuerzo; en contraposición MONEPS necesita inputs (p.e. número de requisitos funcionales del proyecto) fundamentados en la norma referida anteriormente, para estimar el costo y tiempo de desarrollo de un producto software. A través de MONEPS, se muestra el mecanismo predictivo de las RNA's [49] para mejorar la precisión en la tarea de estimar el tiempo y costo para el desarrollo de software.

3 Propuesta del modelo neuronal de estimación para el esfuerzo de desarrollo en proyectos de software (MONEPS)

3.1 Topología neuronal para MONEPS

La estructura neuronal seleccionada para MONEPS es de tipo Backpropagation. Estas redes (ver Figura 3.1) presentan algunas características como [50] [51] [52]:

- Aprenden de manera supervisada e inductiva.
- Son suficientes 3 capas (una de entrada, otra oculta, y una de salida) para las tareas de aprendizaje e identificación de patrones.
- No tienen mayor complejidad estructural ni algorítmica (lo que no sucede p.e. con las topologías recursivas de redes).
- Existe una buena disponibilidad de herramientas automatizadas, tanto libres como propietarias, para el diseño y funcionamiento de estas redes.
- Han sido utilizadas y probadas de manera satisfactoria en varios campos de aplicación (p.e. reconocimiento de imágenes, clasificación de patrones, codificación/decodificación de información, etc.).

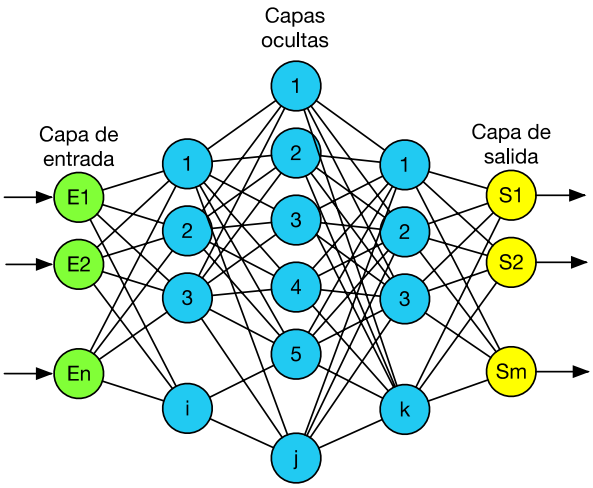

Fig. 3.1 Ejemplo de una red neuronal en cascada (Backpropagation)

3.2 Diseño neuronal de MONEPS

En la Figura 3.2 se muestra un esquema del estándar ISO 25000, compuesto de 6 características y 27 sub características, que permitieron definir la topología de entrada para el modelo neuronal, referido en el presente artículo.

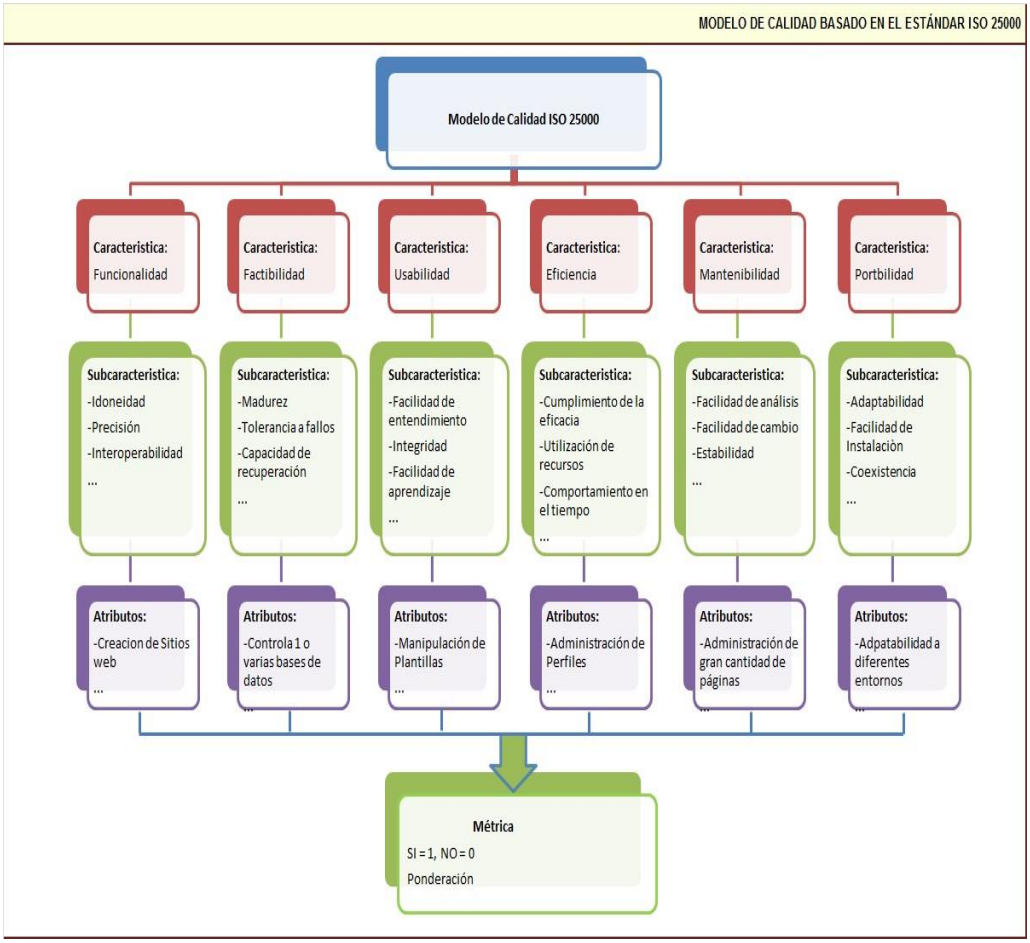

Fig. 3.2 Estructura de la ISO 25000[2]

El diseño de MONEPS considera 42 atributos pertenecientes a características y/o sub características del estándar referido (la Tabla 3.1 describe a modo de ejemplo 5 de estos 42 atributos). Por consiguiente, la capa de entrada tendrá en total 42

[2] Fuente http://iso25000.com/index.php/normas- iso-25000/iso-2501

neuronas, mientras que la capa de salida tendrá solamente 2 neuronas (una para el tiempo y otra para el costo requerido). En la Figura 3.3 se muestra un esquema parcial de la RNA utilizada. Se puede observar en la red las conexiones unidireccionales que van desde la capa de entrada hasta la capa de salida (es decir, no hay ciclos), a través de una capa oculta que tiene 7 neuronas. En la misma figura se muestran sólo unos pocos atributos que forman parte de la RNA usada en MONEPS. Cabe referir que, los atributos pueden ser de naturaleza cuantitativa (p.e. número de programadores asignados) o cualitativa (p.e. experiencia del equipo de desarrollo).

Tabla 3.1 Atributos de ejemplo que forman parte de la capa de entrada para la RNA utilizada por MONEPS

Atributo	Valores	Descripción
Nivel de seguridad	Alto, Medio, Bajo	Indica el nivel de seguridad requerido para la aplicación.
Número de programadores	1, 2, 3, ...	Número de integrantes del equipo de desarrollo asignados al proyecto.
Experiencia del equipo de desarrollo	Alta, Media, Baja	Indica la experiencia del equipo de desarrollo en aplicaciones similares.
Lenguaje de Programación	Imperativo, Declarativo, Orientado a Objetos, Orientado al Problema	Tipo de lenguaje de programación utilizado.
Número de servidores	1, 2, 3, ...	Número de servidores requeridos para la aplicación.

3.3 Resultados obtenidos

Para la fase de entrenamiento de la RNA, fueron utilizados 9 proyectos académicos desarrollados por estudiantes pertenecientes a los últimos niveles de la carrera de Ingeniería en Sistemas e Informática de la Universidad de las Fuerzas Armadas ESPE. De cada proyecto se ingresó la información relacionada con cada uno de los 42 atributos de entrada de la RNA; así por ejemplo, se disponía de información referente a: nivel de seguridad, tiempo de respuesta requerido, nivel de escalabilidad de la aplicación, entre otros. Además, se conocía el tiempo de desarrollo y costo de cada proyecto, lo que posibilitó estructurar las dos salidas de la RNA. El funcionamiento de la RNA es similar a una caja negra que tiene 42 entradas y 2 salidas; dentro de la caja negra la red aprende a configurar patrones de entrada/salida, actualizando los valores de los denominados *pesos sinápticos*[3].

Luego del entrenamiento de la RNA utilizando la herramienta JustNN[4], se pudo constatar que la red tuvo un desempeño satisfactorio después de 42 ciclos (o épocas) de aprendizaje (ver Figura 3.4). El error de la red, durante la fase de entrenamiento

[3] Un peso sináptico representa la conexión o enlace entre dos neuronas.
[4] Las caracteristicas de JustNN están disponibles en: http://www.justnn.com/

estuvo en el orden del 0.5 %, lo que denota un rendimiento muy bueno en relación al mínimo error requerido (target error=1.0%).

En consecuencia, la red aprendió rápidamente a configurar patrones de comportamiento para tiempos y costos referidos a proyectos de software, lo que se puede ver en la Figura 3.5.

La herramienta JustNN permite determinar la importancia relativa de cada atributo de la RNA. Así por ejemplo, se pudo observar que los atributos de la categoría "usabilidad" (norma ISO 25000) tienen un alto impacto en la determinación del esfuerzo para el desarrollo del software; en contraste, la disponibilidad de un arquitecto de software y licencia para bases de datos, tienen poca incidencia en dicho esfuerzo.

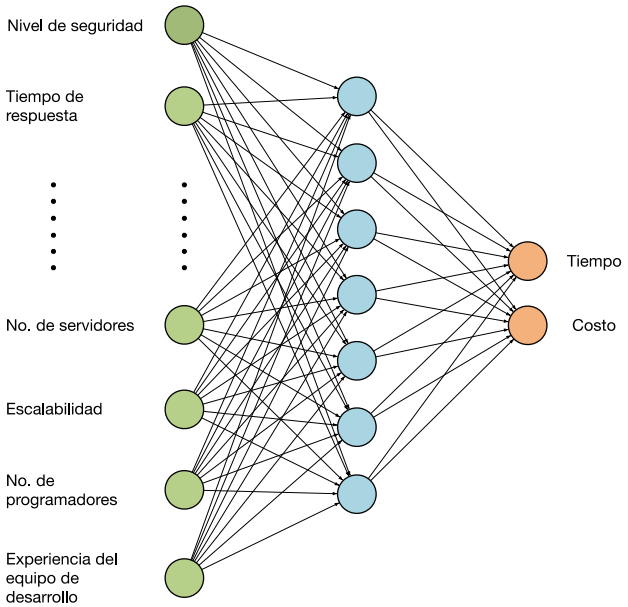

Fig. 3.3 RNA en Backpropagation usada por MONEPS

4 Evaluación del modelo

El desempeño de la RNA que conforma MONEPS, está dado principalmente por su carácter predictivo, es decir, la capacidad de responder ante casos que "no ha visto". En tal virtud, se estudiaron tres proyectos académicos de software, dos de éstos formaron el *conjunto de prueba*[5]. En la fase de evaluación, la RNA fue consultada sobre el tiempo y costo de desarrollo para estos tres proyectos académicos;

[5] Proyectos académicos de software que no fueron considerados en la fase de entrenamiento de la RNA.

previamente se ingresaron (por cada proyecto) 42 datos que definían la naturaleza de cada proyecto académico. Los resultados de esta fase se muestran en la Tabla 4.1.

El primer caso es un simulador para la evaluación de aptitudes de aspirantes para el desarrollo de software, denominado Codesoft. El segundo caso es un sistema de facturación. El tercer caso se refiere a un sistema para control de fichas odontológicas.

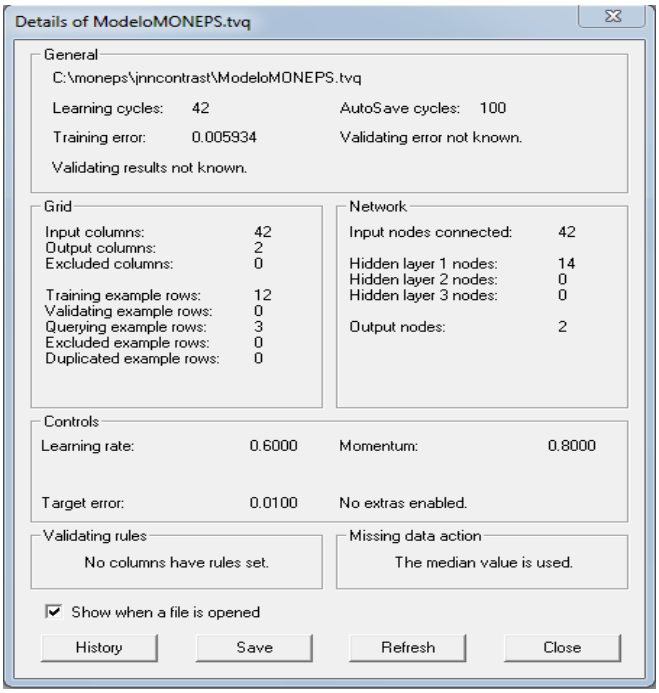

Figura 3.4 Resultados obtenidos durante la fase de entrenamiento de la RNA

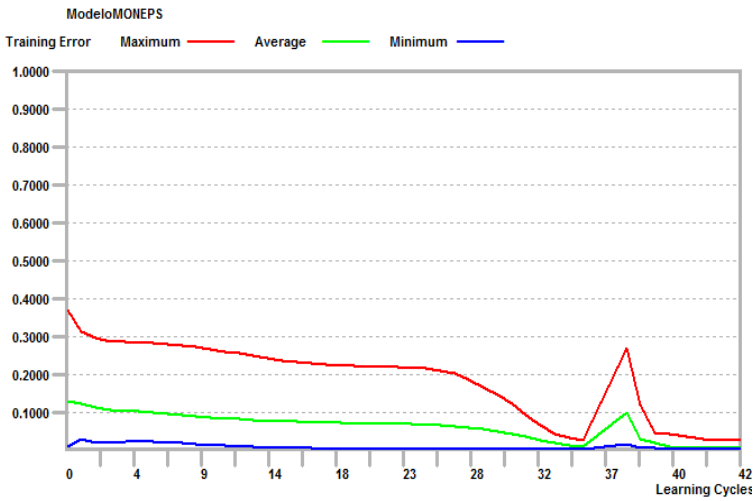

Figura 3.5 Variación del error máximo, mínimo y medio durante el entrenamiento de la red

Tabla 4.1 Tiempo y costo estimado para 2 casos de prueba que la red no ha visto. *También se prueba con un ejemplo de entrenamiento

Caso	Tiempo real de duración (meses)	Tiempo estimado (MONEPS)	Costo referencial (USD)	Costo estimado (MONEPS)
1	3.00	3.65	7558.80	8139.45
2	5.00	3.97	8810.00	7273.05
3*	4.00	3.96	7244.00	7797.52

Tabla 4.2a Tiempos estimados aplicando Cocomo81

Caso	Tiempo (meses)		
	Real	Estimado por Cocomo 81	Estimado por Moneps
1	3.0	6.85	3.65
2	5.0	7.75	3.97

Caso	Nombre del proyecto	Tiempo (meses)			Costo (USD)		
		Real	Estimado por Cocomo-II	Estimado por Moneps	Referen.	Estimado por Cocomo-II	Estimado por Moneps
1	CODESOFT	3.0	12.20	3.65	7558.80	10371.01	8139.45
2	FACTURACIÓN	5.0	10.20	3.97	8810.00	12376.34	7273.05

Caso	Nombre del proyecto	Error relativo para el tiempo		Error relativo para el costo	
		Cocomo-II	Moneps	Cocomo-II	Moneps
1	CODESOFT	306.67%	21.67%	37.20%	7.68%

2	FACTURACIÓN	104.00%	20.60%	40.48%	17.45%

Tabla 4.2b Tiempos y costos estimados en Cocomo-II

Tabla 4.3 Errores relativos en Cocomo-II y Moneps

Puede observarse que, los costos y tiempos estimados por MONEPS, son bastante cercanos a los costos y tiempos reales. Para una mejor contrastación se ha creído conveniente realizar una comparación de resultados con respecto a los modelos Cocomo-81 y Cocomo-II. Dichos resultados se muestran en las tablas 4.2a y 4.2b.

En las tablas mencionadas también se puede apreciar que, MONEPS mantiene una mejor aproximación al tiempo real de duración y costo, con respecto a los modelos Cocomo-81 y Cocomo-II. En la tabla 4.3 se muestra el cálculo del error relativo para Cocomo-II y MONEPS.

5 Conclusiones y trabajo futuro

La RNA utilizada por MONEPS aprendió rápidamente a configurar patrones de comportamiento para tiempos y costos referidos a proyectos de software, y por ende, las estimaciones realizadas son bastante cercanas a los costos y tiempos reales. Los resultados arrojados por la propuesta neuronal, en la fase de evaluación, mostraron mejor precisión respecto a los modelos Cocomo-81 y Cocomo-II, en la estimación de costo y tiempo para proyectos académicos de software. Más específicamente, el error relativo promedio en tiempo y costo, fue respectivamente, 10 y 3 veces menor en MONEPS con relación a Cocomo II, para dos aplicaciones de prueba.

La RNA tiene sus entradas fundamentadas en el uso de atributos del estándar ISO 25000, para la calidad de software. La asociación directa de estos atributos con la capa de entrada de la RNA, ayudó a simplificar la información que alimentó el modelo neuronal. El criterio de diseño para MONEPS, posibilitó la convergencia de aspectos funcionales y no funcionales en la estimación temprana de tiempo y costo para el desarrollo de proyectos de software de tipo académico. Sin embargo, es necesario validar la propuesta neuronal en otros proyectos de software fuera del ámbito académico.

La propuesta denominada MONEPS es de fácil uso y escalable; se pueden realizar los ajustes necesarios para mejorar el nivel de adecuación y precisión en la naturaleza dinámica del software. Tales ajustes se refieren básicamente a activar/desactivar atributos de entrada, y también a la alimentación de la RNA con datos de nuevas aplicaciones.

Como trabajo futuro está la identificación de nuevos atributos críticos y métricas especializadas para los aspectos no funcionales, que podrían incidir notablemente en el nivel de precisión para las estimaciones realizadas por la propuesta neuronal. Asimismo, se pretende más adelante, añadir un componente fuzzy (difuso) para realizar interpretaciones lingüísticas de las entradas y respuestas obtenidas (sistemas neuro-difusos).

Referencias

1. Diaz Villegas, J. E., & Robiolo, G. (2014). Método de estimación de costos de un producto de software web. In XLIII Jornadas Argentinas de Informática e Investigación Operativa (43JAIIO)-XV Simposio Argentino de Ingeniería de Software (Buenos Aires, 2014).
2. Moløkken-Østvold, K., Jørgensen, M., Tanilkan, S. S., Gallis, H., Lien, A. C., & Hove, S. W. (2004, September). A survey on software estimation in the Norwegian industry. In Software Metrics, 2004. Proceedings. 10th International Symposium on (pp. 208-219). IEEE.
3. Omaña, M. (2010). Manufactura Esbelta: una contribución para el desarrollo de software con calidad. Enl@ce: revista Venezolana de Información, Tecnología y Conocimiento, 7(3), 11-26.
4. Pressman, R. (2010). Ingeniería de Software. McGraw-Hill Interamericana de España.
5. Boehm, B. W., Madachy, R., & Steece, B. (2000). Software cost estimation with Cocomo II with Cdrom. Prentice Hall PTR.
6. Boehm, B. W., & Valerdi, R. (2008). Achievements and challenges in cocomo-based software resource estimation. Software, IEEE, 25(5), 74-83.
7. Axelrad, V., Granik, Y., Boksha, V. V., & Rollins, J. G. (1994, September). Cost and yield estimation in a virtual IC factory. In Microelectronic Manufacturing (pp. 102-108). International Society for Optics and Photonics.
8. Madachy, R. (2007). Distributed global development parametric cost modeling. In Software Process Dynamics and Agility (pp. 159-168). Springer Berlin Heidelberg.
9. Jones, C. (1996). How software estimation tools work. American Programmer, 9, 18-27.
10. Madachy, R., & Boehm, B. (2008). Comparative Analysis of COCOMO II, SEER-SEM and True-S Software Cost Models. USC-CSSE-2008-816, University of Southern California Center for Systems and Software Engineering.
11. Southwell, S. V. (1996). Calibration of the Softcost-R Software Cost Model to the Space and Missile Systems Center (SMC) Software Database (SWDB) (No. AFIT/GSM/LAS/96S-6). AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF LOGISTICS AND ACQUISITION MANAGEMENT.
12. Rodríguez, P., Martínez, H. (2010). Modelos Paramétricos de Estimación de Esfuerzo dentro del Proceso de Planificación de Proyectos Software. Revista de Procesos y Métricas, Vol. 7. Asociación Española para la Gobernanza, la Gestión y la Medición de las TI.
13. Ferens, D. V. (1998, July). The conundrum of software estimation models. InAerospace and Electronics Conference, 1998. NAECON 1998. Proceedings of the IEEE 1998 National (pp. 320-328). IEEE.
14. Boehm, B., Abts, C., & Chulani, S. (2000). Software development cost estimation approaches - A survey. Annals of Software Engineering, 10(1-4), 177-205.
15. Webber, B. G. (1995). A Calibration of the Revic Software Cost Estimating Model (No. AFIT/GCA/LAS/95S-13). AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF LOGISTICS AND ACQUISITION MANAGEMENT.
16. Cordero Carrasco, R. J. (2013). Una herramienta de apoyo a la estimación del esfuerzo de desarrollo de software en proyectos pequeños (Doctoral dissertation, Universidad de Chile).
17. Mizell, C., & Malone, L. (2007). A project management approach to using simulation for cost estimation on large, complex software development projects.

18. International Organization for Standardization & International Electrotechnical Commission. (2005). ISO/IEC 25000, Software product Quality Requirements and Evaluation (SQuaRE).
19. Boehm, B. (2000). Safe and simple software cost analysis. IEEE software, (5), 14-17.
20. Baik, J., Boehm, B., & Steece, B. M. (2002). Disaggregating and calibrating the CASE tool variable in COCOMO II. Software Engineering, IEEE Transactions on, 28(11), 1009-1022.
21. López, J. E., & DOLADO COSÍN, J. J. (2008). Estimación del Esfuerzo Software: Factores vinculados a la aplicación a desarrollar. Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos, 2(1).
22. Robiolo, M. G., & Informáticas, C. (2009). Transacciones, Objetos de Entidad y Caminos: métricas de software basadas en casos de uso, que mejoran la estimación temprana de esfuerzo (Doctoral dissertation, Tesis Doctoral UNLP. http://postgrado. info.unlp.edu.ar/Carrera/Doctorado/Tesis%20Doctorales.html).
23. Rodríguez, D., Pytel, P., Tomasello, M., Pollo Cattaneo, M. F., Britos, P. V., & García Martínez, R. (2011). Estudio del Modelo Paramétrico DMCoMo de Estimación de Proyectos de Explotación de Información. In XVII Congreso Argentino de Ciencias de la Computación.
24. Kemerer, C. F. (1987). An empirical validation of software cost estimation models. Communications of the ACM, 30(5), 416-429.
25. Kitchenham, B. A., & Taylor, N. R. (1984). Software cost models. ICL technical journal, 4(1), 73-102.
26. Low, G. C., & Jeffery, D. R. (1990). Function points in the estimation and evaluation of the software process. Software Engineering, IEEE Transactions on, 16(1), 64-71.
27. Hochstetter, J., Diaz, C., & Cares, C. (2012, June). Software call for tenders: Metrics based on speech acts. In Information Systems and Technologies (CISTI), 2012 7th Iberian Conference on (pp. 1-6). IEEE.
28. Rubio, M. G. (2006). Aplicación de técnicas de clustering para la estimación del esfuerzo en la construcción de proyectos software (Doctoral dissertation, Universidad de Alcalá).
29. Chiu, N. H., & Huang, S. J. (2007). The adjusted analogy-based software effort estimation based on similarity distances. Journal of Systems and Software, 80(4), 628-640.
30. Huang, S. J., Chiu, N. H., & Chen, L. W. (2008). Integration of the grey relational analysis with genetic algorithm for software effort estimation. European Journal of Operational Research, 188(3), 898-909.
31. Kumar, K. V., Ravi, V., Carr, M., & Kiran, N. R. (2008). Software development cost estimation using wavelet neural networks. Journal of Systems and Software, 81(11), 1853-1867.
32. Jodpimai, P., Sophatsathit, P., & Lursinsap, C. (2010, March). Estimating software effort with minimum features using neural functional approximation. InComputational Science and Its Applications (ICCSA), 2010 International Conference on (pp. 266-273). IEEE.
33. Garcia, A., Gonzalez, I., Colomo-Palacios, R., Lopez, J. L., & Ruiz, B. (2011). Methodology for software development estimation optimization based on neural networks. Latin America Transactions, IEEE (Revista IEEE America Latina), 9(3), 384-398.
34. Peralta, M. (2004). Estimación del esfuerzo basada en casos de uso. Reportes Técnicos en Ingeniería de Software. Buenos Aires-Argentina, 6(1), 1-16.
35. Páez Anaya, I. D. (2012). Estudio empírico del estado actual de la estimación de software en Pymes de Colombia.

36. Robiolo, G., Castillo, O., Rossi, B., & Santos, S. (2013). Es posible superar la precisión basada en el juicio de expertos de la estimación de esfuerzo de productos de software? X Workshop Latinoamericano de Ingeniería de Software Experimental, ESELAW.
37. Boehm, B. W. (1981). Software engineering economics (Vol. 197). Englewood Cliffs (NJ): Prentice-hall.
38. Putnam, L. H. (1978). A general empirical solution to the macro software sizing and estimating problem. IEEE transactions on Software Engineering, 4(4), 345-361.
39. Galorath, D. D., & Evans, M. W. (2006). Software sizing, estimation, and risk management: when performance is measured performance improves. CRC Press.
40. Vizcaíno, A., García, F., & Piattini, M. (2014). Visión General del Desarrollo Global de Software. International Journal of Systems and Software Engineering for Big Companies.
41. Lopez-Martin, C. (2011). A fuzzy logic model for predicting the development effort of short scale programs based upon two independent variables. Applied Soft Computing, 11(1), 724-732.
42. de Barcelos Tronto, I. F., da Silva, J. D. S., & Sant'Anna, N. (2008). An investigation of artificial neural networks based prediction systems in software project management. Journal of Systems and Software, 81(3), 356-367.
43. Huang, X., Ho, D., Ren, J., & Capretz, L. F. (2007). Improving the COCOMO model using a neuro-fuzzy approach. Applied Soft Computing, 7(1), 29-40.
44. Afzal, W., & Torkar, R. (2011). On the application of genetic programming for software engineering predictive modeling: A systematic review. Expert Systems with Applications, 38(9), 11984-11997.
45. Ruíz Constanten, Y., & Cordero Morales, D. (2013). Estimación en proyectos de software integrando los métodos de Boehm y Humphrey. Revista Cubana de Ciencias Informáticas, 7(3), 23-36.
46. Páez, J. A. (2003). Avances en la toma de decisiones en proyectos de desarrollo de software (Doctoral dissertation, Universidad de Sevilla).
47. Suri, P. K., & Ranjan, P. (2012). Comparative Analysis of Software Effort Estimation Techniques. International Journal of Computer Applications, 48(21).
48. Isasi Viñuela, P., & Galván León, I. M. (2004). Redes de Neuronas Artificiales.Un Enfoque Práctico, Editorial Pearson Educación SA Madrid España.
49. Attarzadeh, I., Mehranzadeh, A., & Barati, A. (2012, July). Proposing an enhanced artificial neural network prediction model to improve the accuracy in software effort estimation. In Computational Intelligence, Communication Systems and Networks (CICSyN), 2012 Fourth International Conference on (pp. 167-172). IEEE.
50. Pino Díez, R., Fuente García, D. D. L., Parreño Fernández, J., & Priore Moreno, P. (2002). Aplicación de redes neuronales artificiales a la previsión de series temporales no estacionarias o no invertibles.
51. Russell, S. J., & Norvig, P. (2009). Artificial Intelligence: A Modern Approach. Third Edition. Prentice Hall.
52. Warwick, K. (2013). Artificial intelligence: the basics. Routledge.