MICHELET DEL CARPIO CHÁVEZ

ESTIMATIVAS DE ESFORÇO DE PROJETOS DE SOFTWARE UTILIZANDO PONTOS DE CASO DE USO E REDES NEURAIS ARTIFICIAIS

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Orientador
Prof. Rudini Menezes Sampaio

Co-orientador
Profª. Ana Cristina Rouiller.

LAVRAS
MINAS GERAIS – BRASIL
2005

MICHELET DEL CARPIO CHÁVEZ

ESTIMATIVAS DE ESFORÇO DE PROJETOS DE SOFTWARE UTILIZANDO PONTOS DE CASO DE USO E REDES NEURAIS ARTIFICIAIS

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Áreas de Concentração:

Engenharia de Software
Inteligência Artificial

Orientador:
Prof. Rudini Menezes Sampaio.

Co-orientador
Profª. Ana Cristina Rouiller.

LAVRAS
MINAS GERAIS – BRASIL
2005

MICHELET DEL CARPIO CHÁVEZ

ESTIMATIVAS DE ESFORÇO DE PROJETOS DE SOFTWARE UTILIZANDO PONTOS DE CASO DE USO E REDES NEURAIS ARTIFICIAIS

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em ____ de Julho de 2005.

_____
Prof. Dr. Guilherme Bastos Alvarenga.

_____
Adler Diniz de Souza.

_____
Profª. Dra. Ana Cristina Rouiller.
(Co-orientadora)

_____
Prof. Dr. Rudini Menezes Sampaio
(Orientador)

LAVRAS
MINAS GERAIS – BRASIL
2005

O homem é apenas seu projeto, só existe na medida em que se realiza, ele é tão-somente o conjunto de seus atos.

Jean Paul Sartre

Agradecimentos

Agradeço à minha família, que mesmo de longe nunca me deixou de apoiar; aos professores que conseguiram iluminar a vida acadêmica de um simples aluno - em especial à professora Ana Cristina; aos meus colegas que conseguiram me suportar na Faculdade e de maneira particular à minha amiga Joelma por ter sido participe de uma amizade *sui generis* durante os anos da graduação.

RESUMO

ESTIMATIVAS DE ESFORÇO DE PROJETOS DE SOFTWARE UTILIZANDO PONTOS DE CASO DE USO E REDES NEURAIS ARTIFICIAIS

Gerenciar esforço em projetos de software é uma área muito importante na gerência de projetos. Pesquisas envolvendo técnicas de inteligência artificial para a produção de estimativas a partir de métricas existentes têm mostrado melhores resultados que métodos estatísticos tradicionais. Este trabalho procura mostrar novas metodologias de estimação de esforço e sua relação com técnicas de aprendizado de máquinas como Redes Neurais.

Palavras-Chave: Pontos de caso de uso, aprendizado de maquinas, estimativas de esforço.

ABSTRACT

Software Project Effort Estimates using Use Case Points and Artificial Neural Networks

Software effort management is a very important discipline in Project Management. Several researches on metric estimates involving Artificial Intelligence methodologies show better results when compared with traditional statistic methods. This work aims to show new methodologies for effort estimation and its relation with machine learning models like Neural Networks.

Keywords: Use Case Points, Machine Learning, effort estimation.

Sumário

1.

LISTA DE TABELAS

2.

LISTA DE FIGURAS

3.

4. INTRODUÇÃO

Em meados dos anos 1990, o *Standish Group* examinou 8.000 projetos de software. Como resultado, foi encontrado que, em média, um projeto excedia seu orçamento em 90% e seu cronograma em 222%. Mais de 50% dos projetos terminados cumpriam menos de 50% de seus requisitos originais [THE STANDISH GROUP (1995)].

Estas estatísticas representam sintomas de problemas mais profundos, que incluem a falta de informação bem no começo do ciclo de vida do projeto. Esta escassez de dados não é surpresa, se considerada a distribuição de organizações de software segundo o modelo *Capability Maturity Model* (CMM) do Instituto de Engenharia de Software [PAULK,M.C et al (1993)]. Um nível CMM abaixo do 3, de 5 possíveis, indica falta de documentação escrita, o que inclui dados, em várias fases do ciclo de vida. Entre 80% e 85% de todas as empresas de software que existem estão abaixo do nível 3 do CMM [CMM (1995)], [CURTIS, B.(1993)].

Aqueles que coletam dados e os usam em processos de estimação não estão dispostos a compartilhar esse valioso conhecimento pelo risco de perder uma vantagem competitiva. Como resultado, a escassez de dados ocorre tanto em quantidade como em qualidade para a maioria das organizações. O que também ajuda a explicar a falta de consistência de estimadores de esforço algorítmicos para as fases iniciais de um projeto.

Boehm [BOEHM B. (1981)] afirma que a exatidão nas estimativas para as primeiras fases do projeto variam em um fator de 4 (i.e entre 25 e 400 por cento). Esta afirmação e sustentada também por Heemstra [HEEMSTRA, F. (1992)].

Recentes pesquisas sobre o uso de aprendizagem de maquinas na Engenharia de Software ([MENDONÇA, M. e N.L. SUNDERHAFT. (1999)], [MENZIES, T. (2001)]) indicam que o uso de modelos evolutivos nesta área é uma técnica madura baseada em ferramentas disponíveis que utilizam algoritmos bem conhecidos como redes neurais ou redes bayesianas [CHULANI, S. et al (1999)], [FENTON, N. E. e NEIL, M. (2000)], [HINTON, G. E. (1992)].

Apesar destas facilidades, a quantidade de ferramental que suporta estas técnicas não é grande (como é mostrado em [MENDONÇA, M. e N.L. SUNDERHAFT. (1999)] e [MENZIES, T. (2001)]). Provavelmente a razão principal é a escassez de dados, o que torna difícil o uso de algoritmos de aprendizado e a formulação de teorias baseadas em

dados para construir ferramentas de suporte à decisão logo no inicio do ciclo de vida do software.

Uma outra razão seria o enfoque dado na estimação do projeto. Métodos tradicionais que utilizam medidas de tamanho como a contagem de linhas de código fonte, do inglês *Lines Of Code* (LOC), considera o software sob a perspectiva da estrutura interna e é mais aplicada nas fases finais do projeto.

4.1. Objetivos

Os objetivos do presente trabalho são os seguintes:

a) Apresentar a problemática do processo de realizar estimativas.
b) Mostrar as tendências atuais na coleta de métricas para um projeto de software.
c) Mostrar por meio de um caso de estudo a eficácia do uso técnicas de Inteligência Artificial no processo de estimativas.

Dentre as várias motivações para a realização desta monografia podem-se ressaltar as seguintes:

a) A necessidade de metodologias eficientes para Estimação de Esforço de Software.
b) A carência de algoritmos adaptados às novas técnicas de estimação (Pontos de Função, Pontos de caso de uso).
c) A eficiência de técnicas de IA de se adaptar onde métodos tradicionais não se mostram adequados

Além deste capítulo introdutório o trabalho possui mais seis capítulos. O Capítulo 2 apresenta alguns conceitos relacionados à medição e às métricas de estimativas de tamanho software utilizando Analise de Pontos de Função (APF) e Pontos de Caso de Uso (PCU), além de apresentar o processo de contagem de Pontos de Caso de Uso e sua relação com as estimativas de esforço. O Capítulo 3 mostra uma visão de geral de redes neurais artificiais, assim como o algoritmo *backpropagation,* como uma técnica de aprendizado de máquina. Este capítulo se justifica porque o estudo de caso exposto neste trabalho utiliza esta técnica de Aprendizado de Máquina. O Capítulo 4 descreve a metodologia utilizada para o presente trabalho, apresenta também uma visão geral sobre Aprendizado de Máquinas e tenta estabelecer uma justificativa para o uso de técnicas de inteligência artificial como

ferramental para auxiliar certos processos de Engenharia de Software, também apresenta Estudo de caso relevante na aplicação de Redes Neurais Artificiais na estimativa de esforço. No Capítulo 5 serão discutidos os resultados do caso de estudo e, finalmente o Capítulo 6 apresenta a conclusão e sugestões de trabalhos futuros.

5. MÉTRICAS DE ESTIMATIVAS DE TAMANHO DE SOFTWARE.

5.1. Introdução

Eficiência, entrega do produto no prazo, dentro do orçamento e com um nível de qualidade desejado pelo cliente são características que influenciam no sucesso de organizações de desenvolvimento de software no mercado atual, globalizado e competitivo de TI. Neste sentido, ressaltam-se a importância de mecanismos de acompanhamento e a avaliação do progresso do processo, projeto e produto.

Estes mecanismos, normalmente baseados em informações qualitativas e quantitativas, coletadas através de medições realizadas durante o projeto, são recursos essenciais na gestão de desenvolvimento de software. Para o gerente do projeto, essas medidas, também conhecidas como métricas, permitem melhor entendimento do processo de produção e do controle do projeto de software. Desta maneira, as medições fornecem informações que permitem a determinação de pontos fortes e fracos dos processos e produto, indicam ações corretivas e propiciam avaliações de impactos de tais ações. Enfim, garantem a qualidade do processo e dos produtos obtidos [FENTON, N. e PFLEEGER, S. (1997)],[BASILI. V.; CALDIERA, G.; ROMBACH, H (1994)].

5.2. Medição de Software.

[FENTON, N. e PFLEEGER, S. (1997)] e [BASILI. et al (1994)],definem medição como um mecanismo para criar memória corporativa e auxiliar nas buscas de respostas para diversas questões associadas ao processo de software. Estes autores ressaltam que para ser efetiva, a medição tem que focar em objetivos específicos, ser aplicada durante todo o processo de desenvolvimento do produto, processo e projeto e ser interpretada com base na caracterização e no entendimento do contexto organizacional.

Por outro lado, [MC GARRY, J et al (2001)] ressaltam que o sucesso do programa de medição depende de um processo estruturado e repetitivo que define as atividades de medidas (coleta, análise e apresentação dos dados) diretamente relacionadas às

necessidades de informação dos gerentes de projetos. Estes autores destacam também, que a medição é mais efetiva quando implementada de forma integrada com as atividades técnicas e de gestão que definem os projetos de software, porque a medição fornece informações objetivas relacionadas aos riscos e problemas que podem ter impacto nos objetivos dos projetos definidos.

[PRESSMAN, R (2002)], com base no "*IEEE Standard Glossary of Software Engineering Terms*" define medida como " uma indicação quantitativa da extensão, dimensão, capacidade ou tamanho de algum atributo de um produto ou de um processo" e métrica como " uma medida quantitativa do grau em que um sistema, componente ou processo possui determinado atributo" . Com base nas medidas, são obtidos os indicadores que são uma métrica, ou combinação de métricas que fornecem compreensão de um processo, recursos e projeto ou produto de software [PRESSMAN, R (2002)].

As métricas podem ser classificadas em várias categorias: métricas de processo, produto e recursos [FENTON, N. e PFLEEGER, S. (1997)], objetivas e subjetivas [MOLLER, K. H. e PAULISH, D.J.(1993)] e diretas e indiretas [FENTON, N. E. e NEIL, M. (1991)]; [PRESSMAN, R (2002)].

Para obter resultados significativos, todas as abordagens de métricas de estimativas requerem alguns dados históricos similares ao projeto proposto incluindo, o domínio da aplicação, o nível de maturidade do processo de desenvolvimento, a experiência da equipe, taxas de produtividade, tipos de tecnologia e de ferramentas utilizadas e o grau de reuso [MC GARRY, J et al (2001)]; [FENTON, N. e PFLEEGER, S. (1997)]

A seguir, serão apresentadas as métricas de estimativa de tamanho de software que, em geral, são baseadas em modelos paramétricos.

5.3. Métricas de Estimativas de Tamanho de Software

"Estimativa de tamanho de software é um processo pelo qual uma pessoa ou um grupo de pessoas estima o tamanho de um produto de software" [McPHEE, (1999)]. O tamanho, geralmente, tem impacto na solução técnica e na gestão do projeto já que, estimativas imprecisas podem levar ao fracasso do projeto [ROSS, M *s.d*].

O tamanho do software significa a quantidade de trabalho a ser executado no desenvolvimento de um projeto. Cada projeto pode ser estimado de acordo com o tamanho físico (que são medidos através da especificação de requisitos, análise, projeto e código); com base nas funções que o usuário obtém, na complexidade do problema que o software irá resolver e no reuso do projeto, que mede o quanto o produto será copiado ou modificado a partir de um outro produto existente [FENTON, N. e PFLEEGER, S. (1997)]; [MCPHEE, C. (1999)].

As primeiras métricas de estimativa de tamanho de software surgiram em 1950/1960 e se basearam no tamanho físico de linhas de código (LOC do original *Lines of Source Code* em inglês) [FENTON, N. E. e NEIL, M. (2000)]

[ROSS, M. (s.d.)] destaca duas vantagens no uso de LOC: i) a possibilidade de fazer a estimativa automaticamente; e ii) a facilidade de uso de dados históricos pois grande parte dos dados de estimativas existentes, foram medidos através de LOC. Já, as desvantagens estão relacionadas à ambigüidade, pois a métrica trata de abstrações não textuais, e a falta de significado da medida para o usuário final. Além destas desvantagens, [JONES, (1994)], ressalta que uma contagem em LOC depende do grau de reuso e da linguagem de programação e pode ser cinco vezes superior a uma outra estimativa, devido às diferenças das técnicas de medição de linhas em branco, linhas de comentário, declaração de dados e linhas de instruções. [FENTON, N. e PFLEEGER, S. (1997)] destacam ainda, que a LOC penaliza programas pequenos e bem projetados, não se adapta às linguagens não procedimentais e é de difícil obtenção na fase inicial de planejamento.

LOC foi uma métrica bastante utilizada até meados de 1970. A partir daí surgiram diversas linguagens de programação e conseqüentemente a necessidade de outras formas de estimar o tamanho de software.

Atualmente, são várias as métricas de estimativa de tamanho existentes e grandes as dificuldades para selecionar a mais apropriada para o tamanho de um projeto de software de uma organização. As principais métricas foram desenvolvidas com base nas funções de software tais como: *Function Point Analysis* ou Análise de Pontos de Função; *Bang, Feature Points*, *Boeing´s ED Function Point*, Mark II, *Use Case Point* ou Pontos de Casos

de Uso,COSMIC Full Function Point. [MACPHEE (1999)], [GARMUS, D. HERRON, D (2000)].

A Análise de Pontos de Função (APF) é uma das primeiras métricas a medir o tamanho do software com alguma precisão, é a mais utilizada no mercado, tornando-se padrão internacional em 2002 através da norma ISO/IEC 20.926 ([DEKKERS, (2003)]; [AGUIAR, (2003)]. Atualmente, o mapeamento da APF para estimativa de projetos de software OO é discutido largamente na literatura.

O Ponto de Casos de Uso (PCU) foi definido para estimar projetos Orientados a Objetos (OO), com base na mesma filosofia da APF e Mark II (a estimativa de tamanho é baseada nas funções do software de acordo com a visão do usuário) e no processo "Objetory " , onde foi desenvolvida a técnica de diagramação para o conceito de casos de uso. Posteriormente, Ivar Jacobson desenvolveu a "*Object Oriented Software Engineering* (OOSE)" , metodologia centrada em Casos de Uso[1], técnica largamente utilizada na indústria para descrever e capturar os requisitos funcionais de software [RIBU, K. (2001)]. Considerando-se que o modelo de casos de uso foi desenvolvido para capturar os requisitos baseados no uso e na visão dos usuários, faz sentido basear a estimativa de tamanho e recursos de projetos de software nos casos de usos [DAMODARAN, M; WASHINGTON, A.]

A continuação serão expostas características principais e os processos de contagem para as AFP e o PCU.

[1] Caso de uso representa as funções do sistema. É uma coleção de interações seqüenciais entre o software em desenvolvimento e seus atores externos relacionados a um objetivo específico. Os casos de uso possuem relacionamentos do tipo incluídos e estendidos ([COCKBURN, A., (2000)], citado por [RIBU, (2001)]). O comportamento comum é executado através de casos de usos incluídos e os opcionais, através dos casos de uso estendidos. O relacionamento tipo 'incluído' é usado para evitar a cópia do mesmo texto em fluxos alternativos de diversos casos de uso. Esses tipos de casos de uso são sempre chamados quando o caso de uso é nomeado num passo de um outro caso de uso. O caso de uso 'estendido' indica que uma instância do caso de uso pode ou não incluir o comportamento especificado no outro caso de uso. Este tipo de relacionamento descreve alternativas ou adições ao cenário principal e são escritas separadamente [RIBU, (2001)]. Não há regras precisas para a contagem dos casos de uso incluídos e estendidos.

5.3.1. Análise de Pontos de Função.

A APF foi desenvolvida em meados da década de 70 e publicada em 1979 por Allan Albrecht na tentativa de minimizar as dificuldades associadas às linhas de código como medida de tamanho de software, e de ajudar na geração de um mecanismo que pudesse prever o esforço associado ao desenvolvimento de software [LONGSTREET, (2002)].

A primeira versão da APF visava tratar dos fatores externos, das características importantes para o usuário, ser aplicada no início do processo de desenvolvimento do produto, ser ligada à produtividade econômica e ser independente do código fonte ou linguagem de programação do software [MCPHEE, C. (1999)].

Em 1984, Allan Albrecht refinou esta versão e, posteriormente, com o aumento do uso da APF, tornou-se necessário definir um guia que interpretasse as regras originais para os novos ambientes. Devido a essa necessidade, em 1986 foi criado o *International Function Group Users Group* (IFPUG). A partir desta data, o IFPUG passou a ser o órgão responsável pela definição das regras de contagem, pelo treinamento e pela certificação dos profissionais interessados na aplicação dessa métrica e divulgação de diversas bases de dados históricas de produtividade da indústria de desenvolvimento de software, disponibilizadas por vários órgãos, entre eles o *International Software Benchmarking Standard Group* (ISBSG). Essas informações possibilitam a estimativa de tempo de desenvolvimento de novos projetos de software e produtividade da equipe com base em estimativas anteriores realizadas através da APF ([GARMUS, D. HERRON, D (2000)]; [IFPUG, (2000)]; [LONGSTREET, (2002)]).

A APF tornou-se a métrica mais usada e estudada na história da engenharia de software e no final de 1993 já havia grupos de usuários em 18 países [JONES, C.(1994)].

A APF visa estabelecer uma medida de "tamanho" do software, em Pontos de Função (PF) (unidade de medida para software assim como a hora é unidade de medida para tempo), através da quantificação das funções implementadas sob o ponto de vista do usuário, num enfoque empresarial e não técnico [LONGSTREET, (2002)]. Essa métrica tem como princípio básico focar na especificação de requisitos para obter estimativas de tempo, custo, esforço e recursos na fase inicial do projeto ou na manutenção de software

independente da tecnologia utilizada para implementação ([FUREY, S.(1997)]; [RIBU, K. (2001)]).

A APF permite uma contagem indicativa no início do desenvolvimento sem conhecer detalhes do modelo de dados. Posteriormente, na fase de projeto, essa contagem passa a ser uma estimativa com maior precisão da complexidade das funções e, ao término do desenvolvimento do software, na implantação, é realizada uma contagem detalhada, obtida a partir do grau de complexidade das funções levantadas no processo funcional, modelo de dados, descrição das telas e relatórios ([IFPUG, (2000)], [LONGSTREET, (2002)]). Em geral, as organizações aplicam a APF como: "um método para determinar o tamanho do pacote de software adquirido; para apoiar a análise da produtividade e qualidade; para estimar os custos, recursos e esforços de projetos de desenvolvimento e manutenção de software" ([HAZAN, C.(1999)]; [GARMUS, D. HERRON, D (2000)])

Além destas finalidades, [LONGSTREET, (2002)] destaca outras utilidades da APF tais como:

a) "definir quando e onde fazer reengenharia; estimar casos de testes; entender o aumento do escopo;
b) ajudar em negociações contratuais;
c) desenvolver um conjunto padrão de métricas; e
d) fazer comparação de software".

Para estimar o tamanho do software de acordo com a AFP, o IFPUG definiu os procedimentos de contagem conforme descritos a seguir.

5.3.1.1. Processo de Contagem de Análise de Pontos de Função

O processo de contagem da APF compõe-se de sete etapas [IFPUG,(2000)]:

i) Determinar o tipo de contagem - A APF se propõe a estimar o tamanho de projetos de desenvolvimento; aplicações em uso ou projetos de manutenção.

ii) Identificar o escopo da contagem e a fronteira da aplicação – O escopo define as funções que serão contadas de acordo com a visão do usuário, e a fronteira da aplicação separa o projeto a ser contado dos usuários e das aplicações externas ao domínio do projeto.

iii) Contar as funções de dados - As funções de dados consistem de: Arquivos Lógicos Internos (ALI's)[2] e Arquivos de Interface Externa_ (AIE's)[3]. Um AIE de uma aplicação sempre será contado como um ALI na aplicação de origem. Para identificar um ALI, observar as seguintes regras:

a. se o grupo de dados ou informações de controle é lógico e identificável pelo usuário;

b. se o grupo de dados é mantido através de um processo elementar dentro da fronteira da aplicação a ser contada.

A complexidade dos ALI's é identificada através do número de itens de dados[4] e de registros lógicos[5]. Para identificar os itens de dados, seguir as seguintes regras:

a. contar um item de dados para cada campo reconhecido pelo usuário como único, não repetido, mantido por um ALI ou recuperado de um AIE ou de um ALI;

b. quando duas aplicações mantêm e ou referenciam o mesmo ALI ou AIE, mas cada uma mantém ou referencia itens de dados separados, conte apenas os itens de dados utilizados em cada aplicação para avaliar a complexidade dos ALI e AIE;

c. contar um item de dado para cada campo requerido pelo usuário para estabelecer um relacionamento com outro ALI ou AIE que, no caso, é considerada chave estrangeira.

Para identificar um AIE, verificar grupos de dados ou informação de controle que satisfaçam à definição de um AIE. As seguintes regras devem ser verificadas:

a. O grupo de dados ou informação de controle é lógico e identificável pelo usuário;

b. o grupo de dados é referenciado pela aplicação a ser contada mas pertence à outra aplicação fora da fronteira da aplicação;

c. o grupo de dados não mantido pela aplicação a ser contada;

d. o grupo de dados é mantido em um Arquivo Lógico Interno de outra aplicação.

---

[2] Arquivo Lógico Interno é um grupo lógico de dados ou informações de controle sob o ponto de vista do usuário, cuja manutenção é feita internamente pela aplicação.

[3] Arquivo de Interface Externa é um grupo lógico de dados referenciado na aplicação cuja responsabilidade pela manutenção é de outra aplicação, ou seja, não é mantido pela aplicação que está sendo contada. São armazenados fora da fronteira da aplicação.

[4] Item de dados representa um segmento de um registro do ALI que tem significado único e é identificado pelo usuário. São os campos da tabela e deve-se contabilizar um item de dado para cada campo.

[5] Registros lógicos são subgrupos de dados reconhecidos pelo usuário dentro de um Arquivo Lógico Interno.

De acordo com o número de itens de dados e de registros lógicos identificados, classificasse o ALI e AIE conforme a Tabela 1 e atribui os pesos de 7, 10 ou 15 PF's para os ALI's e 5, 7 ou 10 PF's para os AIE respectivamente à complexidade baixa, média ou alta. Complexidade Funcional [IFPUG, (2000)]

Tabela 2.1 Complexidade Funcional [IFPUG, (2000)

	ITENS DE DADOS REFERENCIADOS		
REGISTROS LÓGICOS	1 a19	20 a 50	51 ou mais
1	BAIXA	BAIXA	MÉDIA
2 a 5	BAIXA	MÉDIA	ALTA
6 ou mais	MÉDIA	ALTA	ALTA

iv) Contar as funções transacionais - Estas funções representam as funções de processamento dos dados fornecidos pela aplicação ao usuário. As funções transacionais podem ser Entradas Externas (EE)[6], Saídas Externas (SE)[7] e Consultas Externas (CE)[8] Compõem-se de itens referenciados (parâmetros que estão sendo representados pelos seus tipos) e de arquivos referenciados.

Para identificar as EE's, devem ser analisados todos os processos elementares que recebem dados de fora da aplicação e que atualizam um ou mais ALI's de acordo com as seguintes regras:

a. Os dados ou informação de controle devem ser recebidos de fora da fronteira da aplicação;

b. no mínimo, um AIE é mantido, se a entrada de dados pela fronteira não for informação de controle que altera o comportamento do sistema.

[6] Entradas Externas são transações recebidas de fora da fronteira da aplicação que têm como objetivo manter os ALI e ou alterar o comportamento do sistema. Os dados podem ser recebidos de telas de entrada de dados ou diretamente de outros aplicativos.Estes dados podem ser informações de negócios ou informação de controle.

[7] Saídas Externas são as transações que geram dados e os enviam para fora da fronteira da aplicação. Têm que apresentar informação para o usuário através de processamento lógico diferente ou adicional à recuperação de dados ou informação de controle. Ex: dados transferidos para outra aplicação (dados contidos em um ALI e ou AIE, que são formatados e processados para uso em uma outra aplicação); relatórios (se suas saídas usam processamento ou cálculos distintos); dados derivados (dados que resultam de cálculos, fórmulas, ou outras operações sobre dados originais); formatos gráficos; gerador de relatórios (saídas que são desenvolvidas para o usuário por meio de um gerador de relatórios)

[8] Consultas Externas são combinações entre atividades de entrada e saída de dados, ou seja, é uma solicitação enviada para a aplicação, a qual gera uma recuperação correspondente e exibe os dados.

Para contabilizar o item de dado referenciado na transação de EE, considerar as seguintes regras:

a. os itens de dados de acordo com a visão do usuário;

b. contar os itens de dados, que aparecem mais de uma vez em um arquivo por causa da tecnologia ou técnica de implementação apenas uma vez;

c. considerar um item de dado adicional para as teclas de função ou linha(s) de comando que especificam a ação a ser tomada pela EE;

d. contabilizar os campos não informados pelo usuário, mas que são atualizados em um ALI por uma EE;

e. as mensagens de erros, confirmação ou itens de dados adicionais, devem ser consideradas, caso sejam requeridas pelo usuário.

De acordo com o número de itens de dados, ALI e AIE referenciados, classifica-se a EE em baixa, média ou alta, conforme a Tabela 2 abaixo e os respectivos pesos (3, 4 ou 6 PF).

Tabela 2.2 Complexidade de Entrada Externa [IFPUG, (2000)]

	ITENS DE DADOS REFERENCIADOS		
ARQUIVOS REFERERENCIADOS	1 a 4	5 a 15	16 ou mais
0 a 1	BAIXA	BAIXA	MÉDIA
2	BAIXA	MÉDIA	ALTA
3 ou mais	MÉDIA	ALTA	ALTA

Para identificar as SE, verificar o processamento lógico do processo elementar de acordo com as seguintes regras:

a. se contém no mínimo, uma fórmula matemática ou cálculo;

b. se cria dados derivados;

c. se mantém no mínimo, um ALI;

d. se altera o comportamento do sistema.

Já a complexidade da SE é verificada através do número de arquivos e elementos de dados referenciados. Deve-se contar um arquivo referenciado para cada ALI mantido ou lido ou um AIE lido durante o processamento da SE.

Para identificar os itens de dados, observar as regras:

a. contar um item de dado para cada campo reconhecido pelo usuário, não repetido, que entre pela fronteira da aplicação e seja requerido para especificar quando, qual e/ou como o dado será recuperado ou gerado pelo processo elementar ou que saia da fronteira da aplicação. Se um item de dado entra e sai pela fronteira da aplicação, contá-lo apenas uma vez;

b. contar um item de dado quando a aplicação enviar mensagens de resposta para fora da fronteira, indicar um erro ocorrido durante o processamento, confirmar o processamento completo ou verificar se o processamento deva continuar;

c. Contar um item de dado para a habilidade de especificar uma ação a ser tomada, quando existem múltiplos métodos para chamar o mesmo processo lógico;

d. não contar os campos que são recuperados ou derivados pelo sistema e armazenados em um ALI durante o processo elementar se esses campos não cruzam a fronteira da aplicação;

e. não contar variáveis, números de páginas, data/hora ou selos gerados pelo sistema. Após identificar os arquivos e os itens de dados referenciados, classificar a SE conforme a Tabela 3 abaixo e atribuir o peso 4, 5 ou 7 PF respectivo à complexidade baixa, média ou alta.

Tabela 2.3 Complexidade de Saída Externa [IFPUG, (2000)]

	ITENS DE DADOS REFERENCIADOS		
ARQUIVOS REFERENCIADOS	1 a 5	6 a 19	20 ou mais
0 a 1	BAIXA	BAIXA	MÉDIA
2 a 3	BAIXA	MÉDIA	ALTA
4 ou mais	MÉDIA	ALTA	ALTA

A CE visa apresentar informação para o usuário através da recuperação de dados ou informação de controle. Verificar se o processamento lógico é elementar, segundo as regras abaixo:

a. recupera dados ou informação de controle de um ALI ou AIE;

b. não contém fórmulas matemáticas ou cálculos;

c. não cria dados derivados;

d. não mantém ALI;

e. não altera o comportamento do sistema.

De acordo com o número de itens de dados e arquivos referenciados, classificam-se as CE's conforme a Tabela 4 e atribui-se o peso 3, 4 ou 6 PF conforme a complexidade baixa, média ou alta.

Tabela 2.4 Complexidade da consulta Externa [IFPUG, (2000)]

	ITENS DE DADOS REFERENCIADOS		
ARQUIVOS REFERENCIADOS	1 a 5	6 a 19	20 ou mais
0 a 1	BAIXA	BAIXA	MÉDIA
2 a 3	BAIXA	MÉDIA	ALTA
4 ou mais	MÉDIA	ALTA	ALTA

v) Determinar o PF não ajustado - Após identificar as funções de dados e transacionais, multiplicar o total de ALI, AIE, EE, SE e CE pela respectiva complexidade para determinar o valor de PF não ajustado. De acordo com a complexidade, cada uma das funções de dados e transacionais contribui com um determinado número de PF. O total desses pontos de função computados é chamado de pontos de função não-ajustados.

vi) Determinar o fator de ajuste - O nível de influência dos fatores de ajustes técnicos é determinado com base nas 14 características gerais de sistemas, conforme mostra a Tabela 5 e o Apêndice A (questão 1 a 14). Após atribuir e somar os valores dos fatores de ajustes para obter o nível de influência da aplicação, deve-se aplicar a seguinte fórmula:

FATOR DE AJUSTE = (Nível de influência * 0,01) + 0,65

Tabela 2.5 Características gerais do Sistema [IFPUG, (2000)]

CARACTERÍSTICAS GERAIS DO SISTEMA	
Comunicação de dados Processamento distribuído Desempenho Utilização de equipamentos Volume de transações Entrada de dados on-line Eficiência do usuário final	Atualização on-line Processamento complexo Reutilização de código Facilidade de implantação Facilidade operacional Múltiplos locais Facilidade de mudanças

Estas características influenciarão a complexidade do software e conseqüentemente seu tamanho. As características gerais do software podem variar o tamanho do software em 35% mais ou menos.

vii) Calcular Pontos de Função ajustados - Os PF's ajustados são calculados a partir dos PF's não ajustados determinados no ítem 'v' e do fator de ajuste determinado no ítem 'vi' de acordo com a seguinte fórmula:

PF_DESENVOLVIMENTO = PF_NAO_AJUSTADOS * FATOR_AJUSTE

Apesar da APF ser a métrica mais utilizada para estimativa de tamanho de software, essa métrica tem apresentado limitações por ter sido desenvolvida com base nas técnicas estruturadas.

Uma limitação da APF refere-se à tentativa de seu uso na medição do tamanho funcional de outros tipos de software como o de tempo real, científicos e orientados a objetos ([JONES, C. (1994)]; [WHITMIRE, (1992); [REIFER, (1990)]; [MUKHOPADHYAY; KEKRE, (1992)], citados por [ABRAN et al. (2000)]; [RIBU, K. (2001)]

5.3.2. Pontos de Caso de Uso

O PCU é um método de estimativa de tamanho de projeto de software orientado a objetos criado por [KARNER, G. (1993)],com base na APF, Mark II, e no Modelo de casos de uso. O Modelo de casos de uso foi posteriormente incorporado na Unified

Modelling Language (UML) e tem sido muito utilizado no mercado atualmente. O PCU também estima o tamanho da função do software de acordo com a visão do usuário final e com base em uma unidade de medida o PCU.

Neste método, [KARNER, G. (1993)] explora o modelo e a descrição de casos de uso, substitui algumas características técnicas propostas pela APF, cria os fatores ambientais e propõe uma estimativa de produtividade.

Os fatores ambientais e técnicos de ajustes propostos por [KARNER, G. (1993)] foram criados e alterados de acordo com a experiência dos projetos desenvolvidos através do processo "Objectory". [KARNER, G. (1993)] propôs a produtividade de 20 homens/hora por PCU. Com base nos recursos utilizados pelos projetos de seu estudo, tentou ainda encontrar a correlação entre PCU e os recursos necessários para desenvolver o projeto através da adoção da regressão linear, mas seus dados não foram suficientes para encontrar a relação proposta.

O método PCU contribui para a diminuição de algumas dificuldades impostas pelo mercado em relação à resistência de adoção de métodos de estimativas, porque é um método simples, fácil de usar e rápido de se aplicar, quando possui as informações necessárias para realizar as estimativas [DAMODARAN, M; WASHINGTON, A.].

O PCU também possui um processo de contagem conforme descrito a seguir.

5.3.2.1. Processo de contagem de Pontos de Caso de Uso

O processo de contagem de PCU compõe-se de seis etapas [KARNER, G. (1993)]:

i) Contar os atores e atribuir o grau de complexidade- Identificar e multiplicar o total de atores de acordo com o tipo de complexidade (simples, médio ou complexo) pelo respectivo peso (1, 2 ou 3), conforme a Tabela 6, e somar os produtos para obter o total de atores não ajustados.

Tabela 2.6 Peso dos Atores [KARNER, (1993)]

COMPLEXIDADE DO ATOR	DESCRIÇÃO	PESO
Simples	Representa um outro sistema com Interface definida de Programas	1
Médio	Representa um outro sistema que interage através de protocolos ou quando há interação humana através de terminal	2
Complexo	É uma pessoa que interage através de Interface Gráfica ou página Web	3

ii) Contar os casos de uso e atribuir o grau de complexidade - A complexidade é baseada no número de transações. De acordo com o número de transações[9], multiplicar cada caso de uso pelo respectivo peso conforme a Tabela 7

Tabela 2.7 Peso dos casos de uso [KARNER, (1993)]

COMPLEXIDADE DO CASO DE USO	DESCRIÇÃO	PESO
Simples	Tem até 3 transações incluindo os passos alternativos e deve ser realizado com menos de 5 classes de análise	5
Médio	Tem de 4 a 7 transações incluindo os passos alternativos e deve ser realizado com 5 a 10 classes de análise	10
Complexo	Tem acima de 7 transações incluindo os passos alternativos e deve ser realizado com pelo menos de 10 classes de análise	15

[9] De acordo com [Schneider e Winters (2001)], transação é definida como um conjunto de atividades atômicas, as quais são executadas completamente ou não. Quando tiver a mesma transação em todos os diagramas de seqüência, como procedimentos de segurança, a transação deve ser contada apenas uma vez porque a funcionalidade é implementada apenas uma vez e reutilizada em outros casos de uso [RIBU, (2001)].

iii) Calcular os PCU não ajustados - De acordo com a seguinte formula:

$$PCU\_NÃO\_AJUSTADOS = \sum \text{ PESO DOS ATORES} + \sum \text{ PESO CASOS DE USO}$$

iv) Determinar o fator de complexidade técnica (Tabela 8). Os fatores de complexidade técnica variam numa escala de 0 a 5, de acordo com o grau de dificuldade do sistema a ser construído. O valor 0 indica pouca criticalidade baixa complexidade (irrelevante para o projeto) e o valor 5 indica alta criticalidade e complexidade (essencial). Após determinar o valor dos fatores, multiplicar pelo respectivo peso, somar o total e aplicar a seguinte fórmula:

$$FATOR\_DE\_COMPLEXIDADE\_TECNICA = 0{,}6 + (0{,}01 * \sum \text{ FATOR TÉCNICO})$$

Tabela 2.8 Fatores de complexidade técnica [KARNER (1993)]

DESCRIÇÃO	PESO
Sistemas Distribuídos	2,0
Desempenho da aplicação	1,0
Eficiência do usuário final (on-line)	1,0
Processamento interno complexo	1,0
Reusabilidade do código em outras aplicações	1,0
Facilidade de instalação	0,5
Usabilidade (facilidade operacional)	0,5
Portabilidade	2,0
Facilidade de manutenção	1,0
Concorrência	1,0
Características especiais de segurança	1,0
Acesso direto para terceiros	1,0
Facilidades especiais de treinamento	1,0

iv) Determinar a eficiência do fator ambiental - Os fatores ambientais indicam a eficiência do projeto e estão relacionados ao nível de experiência dos profissionais.

Esses fatores (Tabela 9) são determinados através da escala de 0 a 5, onde 0 indica baixa habilidade (pouca experiência no domínio da aplicação), 3 indica média experiência e 5 indica alta experiência. Por exemplo, para o fator “motivação”, 0 significa sem motivação para o projeto; 3, média motivação e 5, alta motivação. Para o fator “requisitos estáveis” , 0 significa extremamente instável; 3, médio e 5, estável. Para o fator “trabalhadores com dedicação parcial”, 0 indica poucos ou nenhum profissional de período não integral; 3, médio e 5 todos profissionais de período não integral. Para o fator “dificuldade da linguagem de programação”, 0 indica que a linguagem é de pouca

complexidade e ou que a equipe de projeto tem domínio completo sobre a mesma; 3, médio e 5, muita dificuldade com a linguagem de programação. Após determinar o valor de cada fator, multiplicar pelo peso e somar o total dos valores. Em seguida, aplicar a seguinte fórmula:

FATOR_DE_COMPLEXIDADE_AMBIENTAL = 1,4 + (- 0,03) * $\sum$ FATOR AMBIENTAL)

Tabela 2.9 Fatores De Complexidade Ambiental [KERNER (1993)]

DESCRIÇÃO	PESO
Familiaridade com o Processo de Desenvolvimento de Software	1,5
Experiência na Aplicação	0,5
Experiência com OO, na Linguagem e na Técnica de Desenvolvimento	1,0
Capacidade do Líder de Projeto	0,5
Motivação	1,0
Requisitos estáveis	2,0
Trabalhadores com dedicação parcial	-1,0
Dificuldade da Linguagem de Programação	-1,0

vi) Calcular os PCU ajustados - Esse cálculo é realizado com base na multiplicação dos PCU não ajustados, na complexidade técnica e na complexidade ambiental através da seguinte fórmula:

PCU = PCU_NÃO_AJUSTADOS * FATOR_DE_COMPLEXIDADE_TECNICA * FATOR_DE_COMPLEXIDADE_AMBIENTAL

5.4. O uso do tamanho do Projeto na estimativa de esforço ou produtividade.

A estimativa de tamanho é essencial no início do projeto, para ajudar o gerente a planejar o desenvolvimento do produto e estimar esforço, cronograma, recursos, custo e outros fatores como a qualidade exigida pelo usuário final. [PUTNAM] citado por [ROSS, M], [MC GARRY, J et al (2001)]; [FENTON, N. e PFLEEGER, S. (1997)], [MCPHEE, C. (1999)]

O *Capability Maturity Model* (CMM) ,nível 2, área chave "Planejamento de projeto de software", estabelece também que as estimativas de esforço, custo, recursos, cronograma e qualidade sejam relacionadas às estimativas de tamanho do projeto. O CMM requer ainda que as estimativas geradas sejam documentadas e usadas no planejamento e controle do projeto [JOHNSON, D.; BRODMAN, J (2000)];[PAULK,M.C et al (1993)]

Com base na estimativa de tamanho, [KARNER, G. (1993)] propôs uma produtividade de 20 homens/hora por PCU. Mas este fator ainda é um ponto que não existe concordância entre diversos pesquisadores. Por exemplo, [SCHNEIDER, G.; WINTERS, J. (2001)] analisando a proposta de Karner, propuseram verificar novamente os fatores ambientais para contar quantos dos fatores estão abaixo da escala 3 e quantos estão acima desta escala. Se o total for 2 ou menos, eles recomendam usar 20 homens/hora por PCU, se o total for 3 ou 4 deve usar 28 homens/ hora por PCU e se e o total for 5 ou mais, o gerente deve mudar o projeto para que os números possam ser ajustados, caso contrário, o projeto terá grandes riscos de falhas.

Em uma outra análise, Spaks, citado por [ANDA, B et al. (2001)], mostrou que o esforço pode variar entre 15 e 30 homens/hora por PCU.

Em relação ao esforço ou produtividade estimada em APF, o ISBSG possui bases de dados históricas que indicam a produtividade de horas por PF de acordo com o tipo de linguagem adotada no desenvolvimento do projeto de software [ISBSG, (2002)].

Além das estimativas iniciais de prazo, custos e esforço, [Pressman (2002)] ressalta que no decorrer do desenvolvimento do projeto, os PFs estimados, são utilizados para estimar a produtividade, a qualidade e outros atributos de software como erros por PF; defeitos por PF; custo por PF e páginas de documentação por PF e PF por homens/mês. Por outro lado, há autores que destacam que a estimativa de tamanho é uma das atividades

mais difíceis de realizar em uma organização de desenvolvimento de software, principalmente quando é realizada no início do projeto [ROSS, M].

6. REDES NEURAIS ARTIFICIAIS

6.1. Conceito de Redes Neurais

Segundo Laurene Fausett, uma rede neural artificial (RNA) é um sistema de processamento de informação que tem certamente características de desempenho em comum com redes neurais biológicas. Redes neurais artificiais têm sido desenvolvidas como generalizações de modelos matemáticos de cognição humana ou biologia neural, baseadas nas suposições de que:

1. O processamento de informação ocorre em elementos simples chamados neurônios;
2. Sinais são passados entre neurônios através de conexões;
3. Cada conexão tem um peso associado, o qual, em uma rede neural típica, multiplica o sinal transmitido;
4. Cada neurônio aplica uma função de ativação (usualmente não linear) a sua entrada para determinar seu sinal de saída.

Uma rede neural é caracterizada por:

1. Seu padrão de conexões entre os neurônios (conhecido como sua arquitetura);
2. Seu método de determinar os pesos nas conexões (conhecido como seu treinamento, ou aprendizagem);
3. Sua função de ativação.

A rede neural consiste de um grande número de elementos de processamento chamados neurônios, unidades, células ou nós. Cada neurônio é conectado com outros neurônios de forma a se comunicar diretamente, cada um com um peso associado.

O peso representa informações sendo usadas pela rede para resolver um problema. Redes neurais podem ser aplicadas a uma grande variedade de problemas, como armazenar e recuperar dados ou padrões, classificar padrões, executar mapeamentos generalizados de padrões de entrada para padrões de saída, agrupar padrões similares, ou encontrar soluções para problemas de otimização.

Cada neurônio tem seu estado interno, chamado de sua ativação ou nível de atividade, o qual é uma função de entrada. Tipicamente um neurônio envia sua ativação

como um sinal para vários neurônios. É importante notar que um neurônio pode enviar apenas um sinal por vez, apesar do sinal ser espalhado para vários neurônios.

Por exemplo, considere um neurônio Y (Figura 3.1) que recebe entrada dos neurônios X_1, X_2 e X_3. As ativações (sinais de saída) desses neurônios são x_1, x_2 e x_3 respectivamente. Os pesos nas conexões de X_1, X_2 e X_3 para Y são w_1, w_2 e w_3, respectivamente. A entrada da rede, y_in, para o neurônio Y, é a soma dos pesos dos neurônios X_1, X_2 e X_3, ou seja,

$$y\_in = w_1x_1 + w_2x_2 + w_3x_3$$

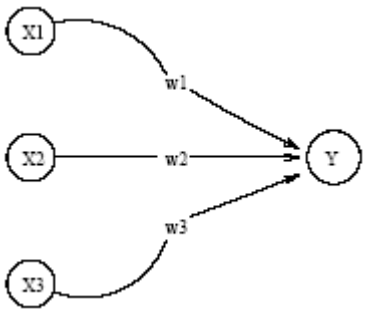


Figura 3.1 Modelo simples de neurônio artificial.

A ativação y do neurônio Y é dada por alguma função de sua entrada, $y = f(y\_in)$, por exemplo, uma função logística binária:

$$f(x) = \frac{1}{1 + \exp(-x)}$$

Suponhamos que o neurônio Y é conectado aos neurônios Z_1 e Z_2 com pesos v_1 e v_2, respectivamente (Figura 3.2). O neurônio Y envia seu sinal y para cada um. Mas em geral, o valor recebido pelo neurônio Z_1 e Z_2 seriam diferentes, porque cada sinal é acompanhado pelo peso apropriado v_1 e v_2. Em uma rede comum, as ativações z_1 e z_2 dos neurônios Z_1 e Z_2 seriam dependentes das entradas de vários neurônios, não apenas um como mostrado no exemplo.

Apesar da rede neural da Figura 3.1 ser muito simples, a presença de uma unidade oculta, junto com uma função de ativação não linear, descreve a sua habilidade para resolver muito mais problemas do que numa rede com apenas unidades de entrada e saída.

Por outro lado, é muito mais difícil de ser treinada (encontrar valores ótimos para os pesos).

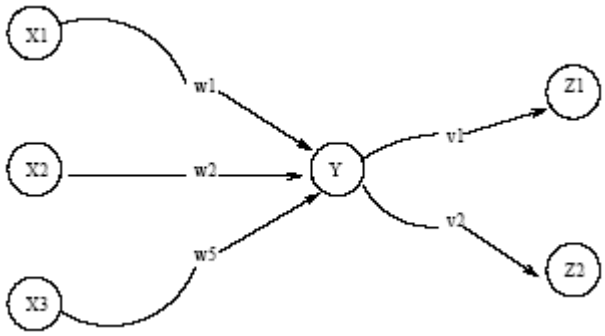


Figura 3.2 Exemplo de rede neural com uma unidade oculta.

6.2. Modelos clássicos de Redes Neurais

McCullock e Pitts, a partir de seu modelo de neurônio (Figura 3.1), mostraram que qualquer função lógica podia ser implementada, através da composição das funções lógicas básicas 'e', 'ou' e 'não'. [McCullock & Pitts, 1943]. Devido a estas afirmações surgiu um grande interesse em modelos de redes neurais que pudessem reproduzir procedimentos mentais, intentando representar o funcionamento dos neurônios biológicos.

6.2.1. O Perceptron

O dispositivo conhecido com o nome de perceptron foi inventado pelo psicólogo Frank Rosenblatt no fim dos anos cinqüenta. Rosenblatt acreditava que a conectividade que se desenvolve nas redes biológicas tem um elevado índice de aleatoriedade. Por isso, desaprovava as análises anteriores, tais como o modelo de McCulloch-Pitts, nos quais se empregava lógica simbólica para analisar umas estruturas bastante idealizadas; e defendia a teoria das probabilidades como a ferramenta de análise mais apropriada.

Os trabalhos de Rosenblatt deram lugar a que se demonstrasse um importante resultado conhecido com o nome de 'teorema da convergência do perceptron'. O teorema afirma, em essência, que se um determinado processo de classificação de padrões pode ser aprendido pelo perceptron, então o procedimento descrito será aprendido em um número finito de ciclos de treinamento.

Em um importante trabalho [Minsky&Papert, 1969] mostraram de maneira rigorosa as diversas limitações computacionais dos modelos existentes na época. Uma delas consistia no fato de que algumas funções muito simples, como a função booleana ou-exclusivo, não podiam ser aprendidas por um perceptron. Isto acontece porque um perceptron é, fundamentalmente, um discriminador lineal, ou seja, pode apenas distinguir, no espaço de suas entradas, entre as duas classes pertencentes a um ou outro dos semi-espaços definidos por um hiperplano, gerado, por sua vez, pelos pesos do perceptron.

6.2.2. Rede Neural *Backpropagation* Padrão

Foi no início da década de 80 que se tornou popular a solução para o aprendizado em *Multi-Layer Perceptron(MLP)*. Este método de aprendizado é chamado *backpropagation* (denominando essas redes de *redes backpropagation*), pois ele se baseia na propagação dos erros a partir da camada de saída (onde eles são facilmente conhecidos devido à saída desejada apresentada) para as camadas anteriores.

Pode-se entender intuitivamente o processo considerando que o erro de uma unidade oculta (ou seja, que não está na camada de saída da rede) é calculado somando-se o quanto ela contribuiu para os erros da camada seguinte. Os erros das unidades da camada seguinte são propagados para esta unidade oculta na mesma medida em que estão ligadas, ou seja, proporcionalmente aos pesos das conexões entre elas. Se a camada seguinte é a camada de saída, seu erro é conhecido; caso contrário, será uma camada oculta cujo erro poderá ser calculado da mesma maneira.

Tecnicamente, este processo é realizado simplesmente alterando o peso de cada conexão na direção que torna o erro menor para o conjunto de exemplos fornecido. Isso é feito calculando-se a derivada do erro para cada peso e alterando-o na direção oposta à derivada. Com isto, há um movimento no espaço dos pesos para a rede em questão sempre em direção a um erro menor, até que se atinja um mínimo local.

Segue-se o algoritmo que resume o processo, em uma versão um pouco mais geral do que a apresentada intuitivamente, que fixa um determinado número de camadas.

Ao invés de ter-se unidades que recebem entradas apenas das unidades das camadas anteriores, teremos uma rede em que as unidades estão em seqüência e cada uma recebe como entradas as ativações de todas as unidades anteriores. Isso inclui as unidades de saída, que recebem como entrada as ativações de todas as unidades, inclusive as das outras unidades de saída anteriores a si. Naturalmente, isso não impede que o esquema por

camadas, mais comum, seja usado, pois ele é a versão particular em que as conexões que não vão de uma camada para outra estão fixadas em zero.

6.2.2.1. O Algoritmo de *Backpropagation*

De acordo com Fausett (1994), o algoritmo *backpropagation* foi criado independentemente por David Parker e LenCun [Parker, 1985] [LenCun, 1986]. O treinamento de uma rede usando esse algoritmo envolve três estágios: o *feedforward* (a passagem para frente) do padrão de treinamento da entrada, o *backpropagation* (a retropropagação) do erro associado e o *ajuste dos pesos*.

Durante o *feedforward*, cada unidade de entrada recebe um sinal de entrada X_i e envia o sinal para cada uma das unidades ocultas $Z_1,...,Z_p$. Cada unidade oculta calcula sua ativação e envia seu sinal z_j para cada uma das unidades de saída. Cada unidade de saída calcula sua ativação y_k para formar a resposta da rede para o dado padrão de entrada.

Durante o treinamento, cada unidade de saída compara sua ativação y_k calculada com o valor esperado t_k para determinar o erro associado com o padrão dessa unidade. Baseado nesse erro, o fator δ_k (k = 1, ..., m) é calculado. δ_k é usado para distribuir o erro na unidade de saída Y_k de volta a todas as unidades da camada anterior (as unidades ocultas que estão conectadas a Y_k). Ele é também usado (depois) para atualizar os pesos entre a camada oculta e a camada de saída. De uma maneira similar, o fator δ_j (j = 1, ..., p) é calculado para cada unidade oculta Z_j. Ele não é necessário para propagar o erro de volta para a camada de entrada, mas é usado para atualizar os pesos entre a camada oculta e a camada de entrada.

Depois dos fatores δ terem sido determinados, os pesos para todas as camadas são ajustados simultaneamente. O ajustamento para o peso w_{jk} (da unidade oculta Z_j para a unidade de saída Y_k) é baseado no fator δ_k e da ativação z_j na unidade oculta Z_j. O ajustamento para o peso v_{ij} (da unidade de entrada X_i para a unidade oculta Z_j) é baseado no fator δ_j e a ativação x_i na unidade de entrada.

6.2.2.2. Nomenclatura

A nomenclatura usada no algoritmo de treinamento para a *rede backpropagation* é a seguinte:

x : Vetor de treinamento de entrada.

$x = (x_1, ..., x_i, ..., x_n)$

t : Vetor de saída esperado.

$t = (t_1, ..., t_k, ..., t_m)$

δ_k : Fator de ajustamento de peso e correção de erro para w_{jk} que é próprio para um erro na unidade de saída Y_k ; também, a informação sobre o erro na unidade Y_k que é propagada de volta à camada oculta que ativa a unidade Y_k .

δ_j : Fator de ajustamento de peso e correção de erro para w_{jk} que de saída para é próprio para a retropropagação da informação do erro da camada a unidade oculta Z_j .

α : Taxa de aprendizagem.

X_i : Unidade de entrada i .

Para cada unidade de entrada, o sinal de entrada e o sinal de saída são os mesmos, nomeado x_i

v_{oj} : *Bias* sobre a unidade oculta j .

Z_j : Unidade oculta j .

A entrada da rede para Z_j é denotada z_in_j :

$$z\_in_j = v_{oj} + \sum_i x_i v_{ij}$$

O sinal de saída (ativação) de Z_j é denotado z_j

$$z_j = f(z\_in_j)$$

w_{ok} : *Bias* sobre a unidade de saída k .

Y_k : Unidade de saída k :

A entrada da rede para Y_k é denotada y_in_k :

$$y\_in_k = w_{ok} + \sum_j z_i w_{jk}$$

O sinal de saída (ativação) de Y_k é denotado y_k :

$y_k = f(y\_in_k)$

6.2.2.3. Função de ativação

Uma função para uma *rede backpropagation* deveria ter várias características importantes: ela deveria ser contínua, diferençável e não decrescente monotonicamente.

Além disso, para eficiência computacional, é desejável que sua derivada seja fácil de computar. Para a função de ativação mais usada comumente, o valor da derivada (num valor particular da variável independente) pode ser expresso em termos do valor da função (no valor da variável independente). Normalmente, é esperado que a função sature, isto é, aproximar de valores máximos finitos e valores mínimos assintoticamente.

Um das funções de ativação mais comum é a função logística binária, a qual tem um intervalo de (0, 1) e é definida como:

$$f_1(x) = \frac{1}{1 + \exp(-x)'}$$

com

$f'(x) = f_1(x)[1 - f_1(x)]$.

6.2.2.4. Pseudo-código do Algoritmo de treinamento *backpropagation*

A função de ativação definida na seção anterior pode ser usada no algoritmo *backpropagation* descrito aqui. A forma dos dados (especialmente os valores dos objetivos) é um fator importante na escolha da função apropriada.

O algoritmo é como se segue:

Passo 0: Inicia os pesos.

(Conjunto de pequenos valores aleatórios).

Passo 1: Enquanto a condição de parada for falsa, faça passos 2 - 9.

Passo 2: Para cada par de treinamento, faça passos 3 - 8.

Feedforward (passe para frente):

Passo 3: Cada unidade de entrada $(X_i, i = 1,...,n)$ recebe sinal de entrada x_i e envia esse sinal para todas as unidades da camada acima (camadas ocultas).

Passo 4: Cada unidade oculta $(Z_j, j = 1,..., p)$ soma seus sinais de peso da entrada,

$$z\_in_j = v_{oj} + \sum_{i=1}^{n} x_i v_{ij}$$

e aplica sua função de ativação para computar seu sinal de saída,

$$z_j = f(z\_in_j)$$

e envia seu sinal para todas as unidades no nível acima (unidades de saída).

Passo 5: Cada unidade de saída $(Y_k, k = 1,...,m)$ soma seus sinais de peso vindos da entrada,

$$y\_in_k = w_{ok} + \sum_{j=1}^{p} z_i w_{jk}$$

e aplica sua função de ativação para computar seu sinal de saída,

$$y_k = f(y\_in_k)$$

Retropropagação (*backpropagation*) do erro:

Passo 6: Cada unidade de saída $(Y_k, k = 1,...,m)$ recebe um padrão de objetivo correspondente ao padrão de treinamento da entrada, calcula seu termo de informação do erro,

$$\delta_k = (t_k - y_k) f'(y\_in_k)$$

calcula seu termo de correção de peso (usado para atualizar w_{jk} depois),

$$\Delta w_{jk} = \alpha \delta_k z_j ,$$

calcula seu termo de correção de *bias* (usado para atualizar w_{ok} posteriormente),

$$\Delta w_{ok} = \alpha \delta_k ,$$

e envia δ_k para as unidades no nível abaixo.

Passo 7: Cada unidade oculta $(Z_j, j = 1,..., p)$ soma seu delta de entrada (de unidades no nível acima),

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$$

multiplica pela derivada de sua função de ativação para calcular seu termo de informação do erro,

$$\delta_j = \delta\_in_j f'(z\_in_j)$$

calcula seu termo de correção de peso (usado para atualizar v_{ij} posteriormente),

$$\Delta v_{ij} = \alpha \delta_j x_i$$

e calcula seu termo de correção de *bias* (usado para atualizar posteriormente),

$$\Delta v_{oj} = \alpha \delta_j .$$

Atualizando pesos e *bias*:

Passo 8: Cada unidade de saída $(Y_k, k = 1,..., p)$ atualiza seu *bias* e pesos $(j = 0,..., p)$:

$$w_{jk}(novo) = w_{jk}(velho) + \Delta w_{jk}$$

Cada unidade oculta $(Z_j, j = 1,..., p)$ atualiza seu *bias* e pesos $(i = 0,..., n)$:

$$v_{ij}(novo) = v_{ij}(velho) + \Delta v_{ij}$$

Passo 9: Teste de condição de parada.

6.2.2.5. Observações importantes

O fato de cada unidade ter como entrada valores de ativação apenas de unidades anteriores é uma restrição importante. Isso permite o cálculo exato das derivadas da função do erro quadrático para um conjunto de treinamento em apenas um passo através da rede, como será mostrado adiante.

A taxa de aprendizado é geralmente um valor pequeno, para que o processo se aproxime da idealização contínua, mas se ela for pequena demais, naturalmente o processo será muito lento. Por isso, seu valor será o maior possível sem que a rede deixe de adaptar seus pesos corretamente. Esta é uma escolha geralmente empírica, que depende de cada caso.

Com este processo, são formadas representações internas úteis nas unidades ocultas da rede, representações necessárias para transformar a entrada de forma que a camada de saída possa separar as classes apresentadas linearmente. A formação destas representações depende em certo grau dos valores iniciais dos pesos das conexões, escolhidos aleatoriamente dentro de uma faixa de valores pequenos.

Muitas vezes estas representações internas são mais eficientes do que inicialmente se supõe.

É interessante notar como, embora o sinal sobre o qual se calcula as derivadas seja normalmente o erro em relação a saídas desejadas, é necessário apenas que se tenha uma medida que se queira minimizar e cuja derivada possa ser calculada em relação aos pesos da rede. Uma aplicação deste princípio já foi usada para forçar a generalização da solução obtida, por exemplo, minimizando uma soma dos erros com uma medida de simplicidade da rede.

Aqui cabem algumas observações sobre os pontos falhos do *backpropagation*. Nem sempre as representações internas necessárias são formadas, devido ao caráter aleatório dos pesos iniciais. Nestes casos, muitas vezes é necessário variar parâmetros como tamanho e taxa de aprendizado de forma *ad hoc*. Considerando o problema em um outro nível, a minimização do erro quadrático no espaço de pesos das conexões possui o inconveniente de todo método de regressão linear: pode ser atingido um mínimo local que não é o global. Além disso, não há indicações a priori sobre qual deva ser o tamanho da rede, nem quantas camadas ela deveria ter (quando usamos uma rede em camadas). Sua aplicação tem, portanto, um forte fator empírico. Ainda assim, o aprendizado por *backpropagation* é bastante eficiente e pode tratar conjuntos complexos de dados. Ele é, de longe, o mais usado em aplicações práticas, quase sempre em classificação de padrões [Braz, 1998].

6.2.2.6. Questões a serem consideradas na implementação do backpropagation

Inicialização dos pesos e *biases*: a escolha dos pesos iniciais é muito importante, pois influencia no quão rápido a rede irá convergir, se ela atingirá um mínimo local ou global da superfície de erro. Como a atualização do peso entre dois neurônios depende da ativação de ambos, é essencial evitar a escolha de pesos iniciais que tornem nula a ativação deles. Segundo Fausett (1994), os pesos iniciais não devem ser valores muito grandes, pois

o valor da derivada da função sigmóide pode ser um valor muito pequeno e o treinamento da rede se torna demorado. Por outro lado, se os pesos iniciais forem valores muito pequenos, a entrada dos neurônios escondidos ou de saída serão valores próximos de zero, o que também implica em um aprendizado muito lento. Geralmente os pesos da rede são inicializados como números aleatórios pequenos. Uma forma é inicializar os pesos como valores aleatórios no intervalo entre -1 e 1 ou -0,5 e 0,5.

As funções de ativação: como mencionado anteriormente, o neurônio gera uma saída aplicando uma função na sua entrada. Geralmente, embora não seja uma regra, a mesma função de ativação é utilizada pelos neurônios da mesma camada. Redes Multi-Camadas requerem funções não-lineares, uma vez que, treinadas utilizando funções lineares, não apresentam nenhuma vantagem em relação às redes de camada única. As funções de ativação mais usadas em *redes backpropagation* são a função logística e a tangente hiperbólica. Isso se deve ao fato de o cálculo da derivada dessas funções ser extremamente simples e não requerer nenhum esforço computacional complementar durante o processo de treinamento.

Função logística binária:

$$f(x) = \frac{1}{1 + \exp(-x)} \qquad f'(x) = f(x)[1 - f(x)]$$

Função logística bipolar:

$$g(x) = \frac{2}{1 + \exp(-x)} - 1 \qquad g'(x) = 0.5[1 + g(x)][1 - g(x)]$$

Por quanto tempo a rede deve ser treinada: No treinamento, a rede deve adquirir a capacidade de memorização e de generalização. Desta forma, ela será capaz de produzir saídas corretas tanto para os padrões de entrada já aprendidos quanto para os nunca vistos anteriormente. Conforme Fausett (1994), para alcançar esse objetivo, a rede deve ser treinada até que o erro quadrático alcance um mínimo. Hecht-Nielsen sugere dois conjuntos de dados durante o treinamento: um de treinamento e outro para testes [HN, 1990]. Estes dois conjuntos são disjuntos. Os ajustes dos pesos são computados utilizando-se os padrões do conjunto de treinamento; entretanto, em intervalos de tempo, o erro é computado para padrões do conjunto de teste. O treinamento continua enquanto o erro para

os padrões de teste diminui. No momento em que esse erro começa a aumentar, a rede está começando a memorizar os padrões de treinamento muito especificamente (perdendo a habilidade de generalizar). Neste ponto, o treinamento termina.

Quantos pares de treinamento são necessários: segundo Fausett (1994), existe uma relação entre o número de pesos da rede W ,o número de padrões de treinamento P , e a taxa de correção da classificação e .

$$e = \frac{W}{P} \qquad P = \frac{W}{e}$$

Ainda conforme Fausett (1994), uma rede treinada para classificar uma fração de $1 - \frac{e}{2}$ padrões de treinamento corretamente, onde $0 < e < \frac{1}{8}$ classificará os padrões de teste corretamente se o número de padrões de treinamento for pelo menos P (P obedece a relação descrita acima). Por exemplo, se $e = 0.1$ e $W = 70$ (rede com setenta pesos), será necessário um conjunto de treinamento com 700 padrões $(P = \frac{70}{0.1})$ para assegurar que 90% dos padrões de teste serão classificados corretamente, assumindo que a rede foi treinada para classificar 95% dos padrões de treinamento corretamente.

A apresentação dos dados: A representação dos dados influencia demasiadamente o desempenho da rede. A velocidade de aprendizado está intimamente relacionada com o formato dos dados de entrada da rede.

Geralmente, os vetores de entrada e saída têm componentes com valores no mesmo intervalo.O cálculo do ajuste de um peso depende da ativação da unidade anterior.

Desta forma, se ativação desta unidade é zero, a rede não irá aprender. Uma sugestão é que os dados estejam na forma bipolar e que a função de ativação seja a função logística bipolar.

Número de camadas escondidas: segundo Fausett (1994), uma camada escondida é suficiente para uma *rede backpropagation* aproximar qualquer mapeamento contínuo de padrões de entrada em padrões de saída, com um certo grau de correção. Entretanto, duas camadas escondidas tornam o treinamento mais fácil e rápido em algumas situações.

Condições de parada: geralmente, o algoritmo pára quando o erro quadrático calculado na saída para todos os padrões de treinamento é inferior ao erro máximo tolerado; ou quando o número de iterações (número de vezes que o algoritmo processa todos os padrões do conjunto de treinamento) excede o número de iterações máximo definido. O tempo também pode ser utilizado para definir a condição de parada da rede. Uma cota de tempo máximo é estabelecida. Quando a rede ultrapassa essa cota, o algoritmo pára.

7. METODOLOGIA

Para realização do presente projeto utilizou-se a pesquisa bibliográfica e a pesquisa documental. Uma breve descrição será feita dessas duas definições na seção 4.1.

7.1. Pesquisa Bibliográfica e Pesquisa Documental

A pesquisa bibliográfica é uma pesquisa desenvolvida a partir de material já elaborado, constituído principalmente de livros e artigos científicos. A pesquisa documental assemelha-se muito à pesquisa bibliográfica. A diferença essencial entre ambas está na natureza das fontes. Enquanto a pesquisa bibliográfica se utiliza fundamentalmente das contribuições dos diversos autores sobre determinado assunto, a pesquisa documental vale-se de materiais que não receberam ainda um tratamento analítico ou que ainda podem ser re-elaborados de acordo com os objetos da pesquisa [Gil, (1991)].

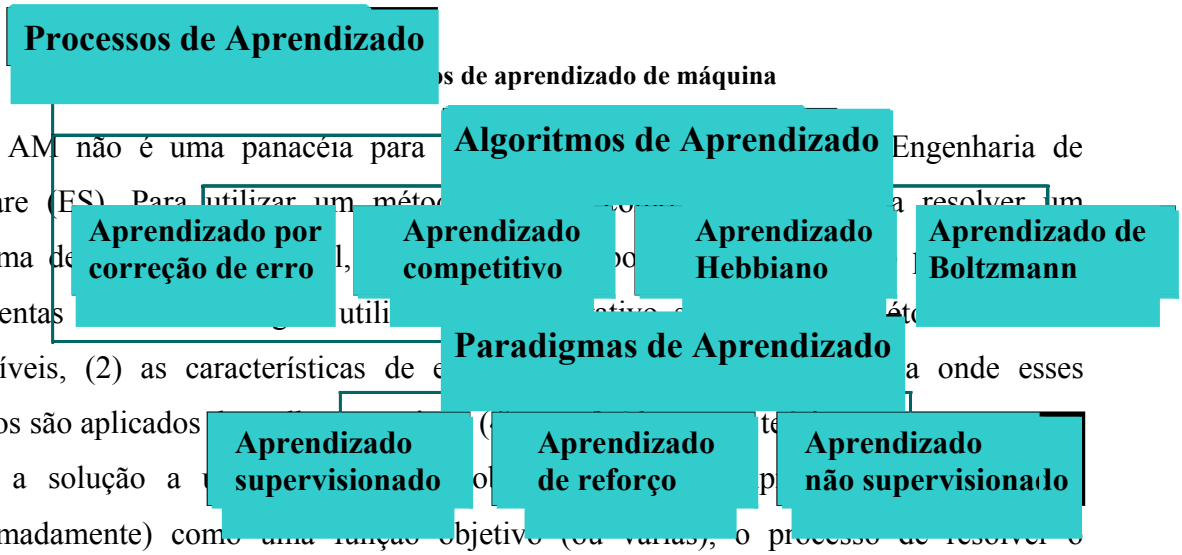
7.2. Procedimento metodológico

Uma pesquisa acerca do uso de métricas de software para estimar esforço foi feita, para possibilitar uma melhor compreensão sobre suas características e a maneira na qual se aplicam em projetos de software. Foi realizada, também, uma revisão minuciosa sobre dados históricos que mostrassem a aplicação de técnicas de aprendizagem de máquinas, como os algoritmos de redes neurais artificiais, utilizadas em estimativas de custo e esforço de software. Aspectos importantes destes análises são apresentados nas seguintes seções.

7.3. Visão geral da Aprendizagem de Máquinas (AM)

Algoritmos de AM tem sido utilizados em diferentes domínios de problemas. Algumas aplicações típicas são problemas que lidam com *data-minning*, onde grandes bancos de dados que contém valiosas regularidades implícitas podem ser descobertas automaticamente ou áreas donde programas devem adaptar-se dinamicamente a condições cambiantes. [MITCHELL, (1997)]. Há uma lista grande de publicações relacionadas com o

estado da prática de aplicações de AM ([BARGADANO, e GUNETTI (1995)], [BRATKO e MUGGLETON (1995], [CRISTIANINI AND SHAWE-TAYLOR, (2000)], [DIETTERICH, (1997)], [MENZIES, (2001)],[MENDONCA M. e N. L SUNDERHAFT,(1999)],[MICHALSKI et al., (1998)],[QUINLAN, (1990)], [SAITTA e NERI, (1998)], [SUTTON e BARTO, (1999)]).



AM não é uma panacéia para Engenharia de Software (ES). Para utilizar um méto a resolver um problema de l, po ferramentas utili ét disponíveis, (2) as características de e a onde esses métodos são aplicados Como a solução a o p aproximadamente) como uma função objetivo (ou várias), o processo de resolver o problema (ou o processo de aprendizagem) se concentra em como encontrar tal função (ou funções) que possam descrever os casos conhecidos e desconhecidos para um o domínio de um dado problema.

Aprender uma função objetivo (ou um conjunto delas) a partir de dados de treinamento está sujeito aos seguintes fatores:

- Representação. Diferentes métodos de aprendizagem podem adotar formalismos especiais para os dados e o conhecimento a ser aprendido. Em alguns casos a função objetivo não é definida explicitamente.

- Características do Processo de Aprendizado. O aprendizado pode ser supervisionado ou não supervisionado. Diferentes métodos podem ter entradas iniciais distintas, técnicas distintas de busca, fatores guia distintos, etc. Dependendo da interação entre o agente que aprende e seu entorno, há um aprendizado de consulta ou um aprendizado de reforço.
- Propriedades dos dados de treinamento e teorias do domínio. Os dados para o processo de aprendizado podem ser pequenos ou grandes em quantidade, exatos ou com 'ruído'. Os resultados de métodos analíticos estarão diretamente relacionados com a completude e a exatidão dos dados, enquanto não estarão ligados diretamente com sua quantidade.
- O retorno da função. Dependendo da saída os problemas de aprendizado podem ser categorizados em: classificação binária, classificação multivalorada e regressão.

[ZHANG, D. ;TSAI, J. (2003)]

7.4. Experiências utilizando Redes Neurais

7.4.1. Descrição geral

Esta seção descreve duas experiências realizadas utilizando redes neurais para criar modelos que consigam explorar melhor ambientes cujos dados são escassos.

Uma rede neural supervisionada pode ser vista como um grafo direcionado composto de nos e conexões (pesos). Um conjunto de vetores, referido como o conjunto de treinamento, é apresentado à rede neural, um vetor por vez. Cada vetor consiste de valores de entrada e valores de saída. O objetivo de uma rede neural é caracterizar a relação entre as entradas e as saídas para todo o conjunto de vetores. Durante o treinamento da rede neural, entradas de um vetor de treinamento se propagam através da rede. As entradas percorrem a rede, enquanto elas são multiplicadas por pesos apropriados e seus produtos são somados. Se a soma excede certo limiar para um nodo, então a saída de esse nodo serve de entrada para um outro nodo. Este valor se repete até que a rede gera um valor de saída para a correspondente entrada do vetor. Este valor calculado é comparado com a saída desejada e um valor de erro é calculado. Dependendo do algoritmo da rede neural, os pesos são reajustados um vetor após outro ou após que toda a rede tenha sido percorrida. De qualquer forma o objetivo é minimizar o valor do erro. Depois de um término satisfatório do treinamento, a arquitetura da rede neural e armazenada e testada com um conjunto independente de vetores chamado conjunto de provas. Se for treinada adequadamente, a rede neural produzirá resultados razoáveis.

Nestes experimentos, utilizar-se-á o enfoque *bottom-up*, no qual os dados provêem do produto, e não do projeto. As entradas da rede neural consistirão em métricas de extraídas a partir do software. Este experimento utilizará LOC como entrada. O valor de saída – esforço de software- representa o número de horas preciso para criar o código fonte. Um vetor no experimento corresponde às métricas extraídas de um programa real. A Tabela 4.1 mostra 104 vetores (programas de computador) consistentes de uma entrada, o tamanho do programa, representado por LOC, e uma saída, representada pelo esforço. Esforço é o número de horas preciso para escrever o programa.

Tabela 4.1 Dados de treinamento para o primeiro experimento

Nro do Vetor	LOC (Entrada)	Esforço (Saída)
1	21	1
	58	1
:	:	:
103	1253	121

Todos os experimentos usam uma rede neural totalmente conectada 1-3-1 (ver figura 4.2), que significa uma entrada, uma saída e uma camada escondida consistente de 3 nos.

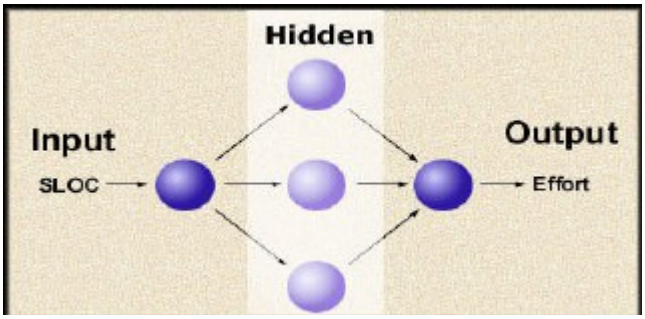


Figura 4.2 Arquitetura 1-3-1 da Rede Neural

Diferentes componentes do modela da rede neural permanecem constantes. *Alpha*, que representa que tão rápido uma rede neural aprende pode variar de zero a um. O valor de *Alpha* é um. *Momentum*, uma variável que ajuda a quebrar um mínimo local, pode também variar de zero a um. O valor do *Momentum* é 1. A função de ativação utilizada é a função sigmóide assimétrica.

Cada passo através dos dados de treino é considerado um *epoch*. Cada experimento foi limitado a um máximo de 1.000 *epoch*s.

O experimento conta com dois conjuntos de dados:

1° Software de Comercio eletrônico: O primeiro set consiste de 104 vetores (unidades de programa); cada vetor contém métricas (ver Tabela 4.2) extraídos de três subsistemas dentro um produto de software.

2° Software de gerencia de frotas: O segundo set de dados consiste em 434 vetores extraídos de uma corporação de software que gerencia frotas.

O código fonte para ambos conjuntos de dados foi feito usando Delphi. Além disso, esses conjuntos de dados são completamente independentes um do outro .

7.4.2. Experimento 1: Software de Comércio Eletrônico.

O primeiro conjunto de experimentos uso os dados da gerência de frotas como dados de treino (434 vetores) e os dados de comercio eletrônico como dados de teste. A Tabela 4.2 ilustra esa configuração.

Tabela 4.2 Dados de treino/teste para o primeiro experimento

	Vetor	LOC	Esforço
Dados de Treino	1	26	1
	:	:	:
	434	4398	245
Dados de Teste	1	15	1
	:	:	:
	104	2796	160

Um experimento consiste em treinar a rede neural por 1.000 *epochs* e grava-la quando a rede treinada gerar os resultados mais precisos contra o conjunto de teste. Geralmente, os melhores resultados acontecem dentro dos primeiros 150 *epochs*.

Foram realizados um total de 10 experimentos. A Tabela 4.3 mostra os resultados do primeiro conjunto de experimentos. Esses resultados mostra o grau de precisão da rede ao estimar o esforço em uma unidade de teste. O fator valor 104 indica o número máximo

possível de acertos. A coluna ‘Corretos” indica o maior número de estimativas acertadas para aqueles vetores no teste. Assim uma estimativa perfeita seria 104. Uma estimativa é considerada correta se está dentro de um 25 por cento do valor real. “Porcentagem Correto” é o resultado da divisão de ‘Corretos’ por ‘Corretos possíveis”.

Tabela 4.3 Resultados para o Experimento 1

Experimento	Acertos possíveis	Acertos	Porcentagem correto
1	104	11	11%
2	104	10	10%
3	104	11	11%
4	104	7	7%
5	104	12	12%
6	104	2	2%
7	104	8	8%
8	104	10	10%
9	104	14	13%
10	104	10	10%

Em média a porcentagem de precisão é de 9%, o qual pode indicar um resultado muito fraco. Mas nas fases iniciais do projeto o interesse é estimar o esforço para o projeto, não para o produto. Para conseguir isto, a partir da perspectiva de projeto, os esforços reais para os produtos são somados e comparados com esforço total de desenvolvimento para o projeto. A Tabela 4.4 mostra os resultados:

Tabela 4.4 Resultados baseados no projeto para Experimento 1

Nro Experimento	Esforço de desenvolvimento para o projeto		Precisão do projeto
	Real	Calculado	
1	2083	1958	-6%
2	2083	1962	-6%
3	2083	1998	-4%
4	2083	2238	7%

5	2083	2110	1%
6	2083	3412	64%
7	2083	2555	23%
8	2083	2104	1%
9	2083	2083	0%
10	2083	1777	-15%

Em média os resultados mostram uma precisão de 90%. A diferencia média entre os dados para os experimentos de 13%.

7.4.3. Experimento 2: Software de Gerência de Frotas.

O experimento 2 usa os dados de comercio eletrônico como o conjunto de dados de treinamento e os dados da gerência de frotas como o conjunto de dados de teste. Basicamente houve uma troca de conjuntos de dados tanto para o treinamento como para o teste. Este experimento queria responder às seguintes questões: era possível treinar a rede com apenas 104 vetores? E: como afetaria a extrapolação de dados na fase de treinamento na rede?

A Tabela 4.5 mostra os resultados do segundo experimento. A coluna um é representa o número real de estimativas calculadas. As colunas dois e três mostram os resultados extrapolados, as colunas quatro e cinco mostram os dados ajustados.

Tabela 4.5 Resultados para os dados de teste para Gerência de Frota

Acertos possíveis	Acertos Reais (Extrapolado)	% corretos (Extrapolado)	Acertos (Ajustado)	% Acertos (Ajustado)
434	130	30%	142	33%
434	133	31%	96	22%
434	78	18%	179	41%
434	118	27%	172	10%
434	132	30%	136	31%
434	130	30%	117	27%
434	134	31%	68	16%
434	146	34%	241	56%
434	130	30%	117	27%

434	106	24%	118	43%

Os resultados são melhores que no primeiro experimento, porém ainda são baixos para calcular estimativas de produto.

Como foi descrito no primeiro conjunto de experimentos, o foco é no esforço de projeto, não no esforço do produto. Uma vez mais, os esforços de cada unidade de produto são somados. A Tabela 4.6 mostra os resultados para o esforço do projeto.

Tabela 4.6 Resultados para o experimento 2 focados no projeto

Esforço real de desenvolvimento do projeto	Esforço calculado (extrapolado)	Precisão do projeto (extrapolado)	Esforço calculado (ajustado)	Precisão do projeto (ajustado)
15949	9464	-41%	14887	-7%
15949	8787	-45%	13821	-13%
15949	9066	-43%	14261	-11%
15949	9809	-38%	15429	-3%
15949	9281	-42%	14599	-8%
15949	8753	-45%	13768	-14%
15949	8640	-46%	13591	-15%
15949	10855	-32%	17074	7%
15949	8915	-44%	14022	-12%
15949	9299	-42%	14627	-8%

A primeira coluna representa o esforço total para o projeto de gerencia de frotas. A coluna dois mostra a soma das estimativas calculadas. O problema de extrapolação é evidente pelo fato de que todos as estimativas sub-estimam o esforço do projeto real. A coluna três mostra a porcentagem de sub-estimação.

A coluna quatro ajusta os resultados da coluna dois. A coluna cinco ilustra a o quanto estão próximos os valores estimados com os valores reais.

Ambos conjuntos mostram que é factível estimar esforço de projeto utilizando técnicas de aprendizado de maquina desde uma perspectiva orientada ao produto, mesmo que fazendo os ajustes necessários.

8. DISCUSSÕES E TRABALHOS FUTUROS

Os experimentos apresentados mostram que é possível somar e dimensionar um conjunto de unidades de programas para atingir uma estimativa do esforço total necessário para um projeto. Mas existe um questionamento importante: como incorporar o esforço de programação dentro da estimativa de projeto logo no inicio do ciclo de vida sem ficar

preso a dados históricos? Uma alternativa que é utilizada atualmente na industria é o uso de ferramental para métricas mais atualizado, tais como o uso de Pontos de Caso de Uso, técnica que logo no início do ciclo de vida do projeto fornece uma visão geral sobre o tamanho do projeto que recém se inicia.

O uso simultâneo de Pontos de Caso de Uso e técnicas de aprendizado de máquinas não foi encontrado na literatura, embora existam diversos estudos que mostram resultados satisfatórios utilizando análise de pontos de função e lógica *fuzzy* ou mesmo redes neurais.

Este cenário representa o estado da arte dos processos de estimativas, mostrando a análise de pontos por função como uma técnica sólida, utilizada por uma fatia importante das empresas de software que utilizam métricas nos seus processos de software e o modelo de pontos de casos de uso como uma técnica derivada, ajustada às necessidades atuais de programação orientada a objetos que ainda não conseguiu situar-se no mercado.

A partir da informação resgatada nesta monografia podem surgir algumas iniciativas para trabalhos futuros, como por exemplo:

- Implementação dos métodos mostrados e a aplicação de outros métodos de aprendizagem de máquina no processo de estimativas como lógica *fuzzy* ou redes bayesianas.
- Criação de um software que agrupe os métodos propostos.

9. CONCLUSÃO

No presente trabalho foram mostradas as metodologias mais recentes para a estimação de tamanho de projetos e de esforço, assim como os aspectos mais relevantes do uso de técnicas de aprendizado de máquinas como ferramental para sua utilização com a engenharia de software, entre elas as Redes Neurais.

No caso de estudo descrito foram mostrados resultados alentadores do uso de redes neurais em processos de estimativas, embora não tão atuais, porém que demonstram a flexibilidade que esta técnica de Inteligência Artificial pode oferecer.

10. REFERÊNCIAS BIBLIOGRÁFICAS

[ABRAN, A. et al (2000)]. Abran, A. et al Functional size measurement methods: COSMIC-FFP: design and field trials. In: FESMA-AEMES Software Measurements Conference. 2000.

[AGUIAR, M. (2003)] Aguiar, M. Pontos de função ou pontos de caso de uso? Como estimar projetos orientados a objetos. Developer´s Magazine V. 7, n. 77. Jan., 2003.

[ANDA, B et al. (2001)] Estimating software development effort based on use cases: experiences from industry. In: International Conference on the Unified Modeling Language (UML2001), 4th. Proceedings Toronto, Oct. 1 – 5, 2001, p. 487 – 502.

[BARGADANO, and GUNETTI (1995)] Bergadano, F. and Gunetti, D. 1995. Inductive Logic Programming: From Machine Learning to Software Engineering, MIT Press.

[BASILI. V. et al (1994)]. Basili. V.; Caldiera, G.; Rombach, H.Goal Question Metric paradigm. In: Encyclopédia of Software Engineering. V. 2, 1994. p. 527 –532

[BRATKO, I and MUGGLETON, S. (1995)] Bratko, I. and Muggleton, S. 1995. Applications of inductive logic programming, Communications of ACM 38(11): 65–70.

[BRAZ, (1998)] Braz, Rodrigo de Salvo. "Alto Nível em Redes Neurais". 1998. 87 págs. Dissertação - Instituto Militar de Engenharia.

[BOEHM, B. (1981)] Boehm, B., Software Engineering Economics, Englewood Cliffs, NJ, Prentice-Hall, 1981.

[CHULANI, S. et al (1999)] Chulani, S., and Boehm, B., and B. Steece, "Bayesian Analysis of Empirical Software Engineering Cost Models", IEEE Transaction on Software Engineering, 25 4, July/August, 1999.

[CMM (1995)] Carnegie Mellon Software Engineering Institute, The Capability Maturity Model: Guidelines for Improving Software Process, Addison-Wesley, Reading, Mass.,1995.

[CRISTIANINI, N. and SHAVE-TAYLOR, J. (2000)] Cristianini, N. and Shawe-Taylor, J. 2000. An Introduction to Support Vector Machines, Cambridge University Press.

[CURTIS, B. (1993)] Curtis, B., Personal Conversation, International Conference on Software Engineering, Baltimore, Maryland, May, 1993.

[DAMODARAN, M; WASHINGTON, A.], Damodaran, M. Washington, A A. Estimation using use case points. Computer Science Program Texas –Victoria: University of Houston. S.d. 4 p. Disponível em: http://bfpug.com.br/Artigos/UCP/Damodaran-Estimation_Using_Use_Case_Points.pdf Acesso em 07/06/2005.

[DEKKERS, C. (2003)] Dekkers, C. Measuring the 'logical' or 'functional' size of software projects and software application". ,ISO BULLETIN May, 2003. p. 10 – 13.

[DIETTERICH T. G. (1997)] Dietterich, T.G. 1997. Machine learning research: four current directions, AI Magazine 18(4): 97–136.

[FAUSSET, (1994)] Fausset, Laurene V. "Fundamentals of neural networks: architectures algorithms, and aplications". 1994.

[FENTON, N. E. e NEIL, M. (1991)] Fenton, N.E., and Neil M, "Software Metrics: A rigorous Approach" Chapman and Hall: 1991.

[FENTON, N. E. e NEIL, M. (2000)] Fenton, N.E., and Neil M, "Software Metrics: Roadmap", The Future of Software Engineering (Ed. Anthony Finkelstein) 22nd International Conference on Software Engineering, ACM Press ISBN 1-58113- 253-0, 2000, Pp 357-370.

[FENTON, N. e PFLEEGER, S. (1997)], Fenton, N. E. e Pfleeger, S. "Software Metrics" a rigorous & practical approach. Boston: PWS Publishing Company, 1997. 638 p

[FUREY, S.(1997)] Furey, S. Why we should use Function Points. ,IEEE Mar./April, 1997. 2 p.

[GIL, A.C. (1991)] Gil, A. C. Como Elaborar Projetos de Pesquisa. 3ª Ed. São Paulo: Editora Atlas S.A, 1991. p. 48 – 51.

[GARMUS, D. HERRON, D (2000)] Garmus, D., Herron, D. Function Point Analysis Measurement Practices for successful software projects. Addison-Wesley: EUA. 2000. 363 p.

[HAZAN, C.(1999)] Hazan, C "Análise de Pontos por Função: uma abordagem gerencial". Rio de Janeiro, SERPRO. 1999. 1 v

[HN, (1990)] Hecht - Nielsen, R. "Neurocomputing". Reading, MA: Addison -Wesley. 1990.

[HEEMSTRA, F. (1992)]Heemstra, F. "Software Cost Estimation," Information and Software Technology, October 1992, Pp. 627-639.

[HINTON, G. E. (1992)] Hinton, G.E., "How Neural Networks Learn from Experience," Scientific American, September, 1992, Pp. 144-151.

[IFPUG, (2000)] International Function Point Users Group. Function Point Counting Practices Case Study 3 – Analysis, Construction. Release 2.0. Princeton Junction: IFPUG. 2001. 246 p.

[JOHNSON, D.; BRODMAN, J (2000)] Johnson, D. Brodman, J. Applying CMM project planning practices to diverse environments. IEEE Software. July/Aug. 2000. p. 40 – 47.

[JONES, C.(1994)] Jones, C "Software Challenges: Function Point a new way of looking at tools. Computer," August, 1994. p. 66 – 67. Acesso em 24/02/03.Disponível em: http://dlib2.computer.org/co/books/co1995/pdf/rx102.pdf

[KARNER, G. (1993)] Karner, G. Use Case Points resource estimation for Objectory projects. Objective Systems SF AB (copyright owned by Rational/IBM), 1993

[LE CUN, (1986)], LenCun Y. "Learning Processes in an Asymmetric Threshold Network". In E. Bienenstock, F. Fogelman-Souli, & G. Weisbuch, eds.Disordered Systems and Biological Organization. NATO ASI Series, F20, Berlin: Springer-Verlag.

[LONGSTREET, D. (2002)] LongStreet, D. “Fundamentals of Function Point Analysis”. Blue Springs: Longstreet Consulting Inc., 2002. Disponível em: http://www.ifpug.com/Articles/default.htm.
Acesso em 01/04/05.

[MCCULLOCH & PITTS, (1943)] McCulloch,W. S., & W. Pitts. (1943). "A Logical Calculus of the Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics.

[MC GARRY, J et al (2001)] Practical Software Measurement: objective information for decision makers. Boston: Addison-Wesley. 2001. 277p.

[MCPHEE, C. (1999)] McPHEE, C. “SENG 621: Software process management:_software size estimation”. University of Calgary. 1999. 11p. Disponível em: http://sern.ucalgary.ca/~cmcphee/SENG621/Software_Size_Estimation.html Acesso em 12/11/2004.

[MENDONÇA, M. e N.L. SUNDERHAFT. (1999)] Mendonça, M, and N.L. Sunderhaft, “Mining Software Engineering Data: A Survey,” Data & Analysis Center for Software, 1999.

[MENZIES, T. (2001)] Menzies, T., “Practical Machine Learning for Software Engineering and Knowledge Engineering,” Handbook of Software Engineering and Knowledge Engineering, 2001. Available from http://tim.menzies.com/pdf/00ml.pdf.

[MICHALSKI, R. S.et al (1998)] Michalski, R.S., Bratko, I., and Kubat, M. (eds.). 1998. Machine Learning and Data Mining: Methods and Applications, John Wiley & Sons Ltd., New York.

[MITCHEL, T(1997a)], Mitchell, T. 1997a. Machine Learning, McGraw-Hill, New York.

[MOLLER, K. H. e PAULISH, D.J.(1993)] MÖLLER, K.H.; PAULISH, D.J.; “Software Metrics : a practioner's guide to improved product development” . IEEE Computer Society Press, Chapmam & Hall; 1993

[MINSKY&PAPERT, (1969)] Minsky, M. L., & S. A. Papert. (1988). "Perceptrons" Expanded Edition. Cambridge, MA: MIT Press. Original edition, 1969.

[PAULK,M.C et al (1993)] Paulk, M.C., and B. Curtis and M.B. Chrissis and C.V. Weber, “Capability Maturity Model, Version 1.1,” IEEE Software, 10 4, July, 1993, Pp. 18-27.

[PARKER, (1985)] Parker, D. "Learning Logic". Technical Report TR-87, Cambridge, MA: Center for Computational Research in Economics and Management Science, MIT.

[PRESSMAN, R (2002)] PRESSMAN, R. Engenharia de Software 5. ed._Rio de Janeiro: McGraw-Hill, 2002. 843 p.

[QUINLAN, (1990)] Quinlan, J.R. 1990. Learning logical definitions from relations, Machine Learning 5(3): 239–266.

[RIBU, K. (2001)] Ribu, K. “Estimating object – oriented software projects with use cases”Oslo: University of Oslo, 2001.132 p. Tese Mestrado–Department of Informatics

[ROSS, M] Ross, M Size does matter: continuous size estimating and tracking. Quantitative Software Management Disponível em: http://www.qsm.com. Acesso em 19/06/05. s.d . 16 p.

[SAITTA and NERI (1998)] Saitta, L. and Neri, F. 1998. Learning in the ‘real world’, Machine Learning 30(2/3): 133–163.

[SCHNEIDER, G.; WINTERS, J. (2001)] Schneider, G. e Winters, J.Use case and project plan. Applying Use cases: a practical guide. 2. ed. New York: Addison Wesley, 2001. capítulo 10, p. 143 – 159.

[SUTTON, and BARTO (1999)] Sutton, R. and Barto, A. 1999. Reinforcement Learning: An Introduction, MIT Press.

[THE STANDISH GROUP (1995)]The Standish Group, CHAOS Chronicles, Standish Group Internal Report, 1995.

[ZHANG, D. ;TSAI, J. (2003)] Zhang, D., Tsai, J. Machine Learning and Software Engineering Software Quality Journal, 11, 87–119, 2003

RESUMO ESTENDIDO

Gerenciar esforço em projetos de software é uma área muito importante na gerência de projetos. As técnicas atuais para gerar estimativas de esforço estão se adaptando continuamente com os novos processos de desenvolvimento e seu grau de precisão está caca vez mais relacionado com o sucesso do projeto. Pesquisas envolvendo técnicas de inteligência artificial para a produção de estimativas a partir de métricas existentes têm mostrado melhores resultados que métodos estatísticos tradicionais e vão conseguindo, empiricamente, bases amplas que suportam a confiança nesses métodos. Este trabalho procura mostrar novas metodologias de estimação de esforço e sua relação com técnicas de aprendizado de máquinas e a importância do uso de técnicas de inteligência artificial no apoio à decisão em tarefas da gerencia de projetos.