# Capítulo 2

**Related Work**

Após uma revisão sistemática da literatura, foram identificadas inúmeras abordagens para análise de risco: Modelo de Regressão Logística (MRL), Rede de Crenças Bayesiana (RCB), Redes Neurais Artificiais (RNA), Análise de Discriminantes (AD), Árvore de Decisão (ADE), Algoritmos Genéticos (AG), Otimização por Enxame de Partículas (PSO), Teoria dos Conjuntos Fuzzy (TCF), Sistemas Neuro-Fuzzy (SNF), Mapas Cognitivos Fuzzy Estendidos (E-FCM) \cite{mizuno2001prediction} \cite{huang2004neuro} \cite{hu2007software} \cite{attarzadeh2010novel} \cite{dzega2010classification} \cite{yu2011software} \cite{saxena2012software} \cite{dan2013improving}.

Hu et al. \cite{hu2007software} propuseram um método para análise de riscos de software e previsão do resultado de projetos de software. Algoritmos Genéticos foram utilizados como um método de otimização de desempenho de Rede Neurais por meio ~~de muitas maneiras, em termos~~ da seleção dos pesos, da estrutura da rede e da regra de aprendizado. A RNA padrão, nesse estudo, teve uma melhoria no desempenho após a inclusão de AG. Os resultados dos experimentos mostraram que após a introdução de AG para o processo de treinamento da RNA, o modelo de avaliação de risco de software modificado pôde ser sensivelmente melhorado ~~notavelmente~~ alcançando uma maior precisão quando comparado com o modelo SVM. Zhang Dan \cite{dan2013improving} propôs um modelo de previsão baseado em RNA que utiliza o Modelo de Custo Construtivo (COCOMO) que foi aprimorado ~~melhorado~~ após a aplicação do PSO, para prover um método de estimativa do esforço no desenvolvimento de software de forma precisa. Attarzadeh e Ow \cite{attarzadeh2010novel} utilizaram a RNA para melhorar a precisão da estimativa do esforço comparado com o modelo tradicional COCOMO. Huang et al. \cite{huang2004neuro} apresentaram um \textit{framework} genérico para a estimativa de software baseado em SNF e melhoraram a estimativa de custo para o COCOMO'81.

Yu \cite{yu2011software} apresentou um modelo baseado na Teoria Fuzzy. Ele superou a dificuldade da avaliação de indicadores qualitativos e quantitativos nos métodos de análise tradicionais. Além disso, Saxena e Singh \cite{saxena2012software} exploraram técnicas Neuro-Fuzzy no projeto de um modelo adequado na utilização de uma melhor estimativa de esforço no desenvolvimento de software para projetos da NASA. Os resultados mostraram que o SNF tem o menor erro de previsão quando comparado com modelos existentes. Por outro lado ~~Enquanto isso~~, Lazzerini e Mkrtchyan \cite{lazzerini2011analyzing} sugeriram um \textit{framework} para análise de riscos usando E-FCM e E-FCM Estendidos, pela introdução de uma representação gráfica especial para análise de risco.

Mizuno et al. \cite{mizuno2001prediction} propuseram um novo método de previsão para projetos de software arriscados. Os autores utilizaram o modelo de regressão logística para estimar se um projeto pode tornar-se arriscado ou não. No entanto, a abordagem de previsão proposta para o custo e a duração ~~definitivamente~~ não tem um nível alto de precisão.

Dzega e Pietruszkiewicz \cite{dzega2010classification} apresentaram resultados de experimentos de análise de riscos realizados usando classificadores de mineração de dados tais como C4.5, RandomTree e Árvore de Regressão e Classificação (CART). Além disso, eles descreveram como os metaclassificadores \textit{boosting} e \textit{bagging} foram aplicados para melhoria dos resultados e também analisaram a influência de seus parâmetros em habilidades de generalização para a precisão da estimativa. Devido a um grande número de atributos desordenados na base de dados, MLP e SVM foram rejeitados prematuramente, gerando baixa precisão para cada conjunto de dados.

Em resumo, alguns desses estudos propuseram métodos para a estimativa de custo, cronograma e esforço de projetos de software; outros estudos propuseram abordagens para classificação de risco e de projetos de software (sucesso, desafiado ou falho); os demais apresentaram técnicas para estimativa do impacto do risco na gestão de projetos de software \cite{yu2011software} \cite{saxena2012software} \cite{lazzerini2011analyzing} \cite{dzega2010classification}.

**Project Risk Management**

Risco pode ser definido como um evento incerto ou condição que, em caso de ocorrência, afeta pelo menos um dos objetivos do projeto. Ele pode ser considerado tanto como uma ameaça (impacto negativo) quanto uma oportunidade (impacto positivo) \cite{PMBOK2008},.

O Gerenciamento de Riscos envolve processos de planejamento, identificação, avaliação e priorização dos mesmos. Numa definição mais apropriada, gerenciamento de riscos pode ser definido como o processo de análise do ~~analisar o~~ grau de exposição ao risco e na determinação ~~determinar~~ de como melhor lidar com essa exposição. Esta área objetiva  ~~Um objetivo dessa área~~ não somente a identificação, como também o desenvolvimento de uma abordagem robusta para gerenciamento de um modo proativo do ~~proativamente o~~ impacto dos riscos no projeto \cite{OSUNDAHUNSI2012}.

De acordo com o PMBOK \cite{PMBOK2008}, o gerenciamento de riscos em projetos envolvem processos relativos ao planejamento, identificação, análise, planejamento de respostas, monitoramento e controle de riscos de um projeto. Seu objetivo é aumentar a probabilidade e o impacto dos eventos positivos e reduzir a probabilidade e severidade dos eventos negativos. Do ponto de vista de gerenciamento, a tomada de decisões conscientes ~~informadas conscientemente~~ pela avaliação do ~~avaliando o~~ que pode dar errado, assim como a probabilidade e a gravidade do impacto, é o cerne do gerenciamento de riscos. Essa atividade envolve a avaliação das vantagens e desvantagens associadas a todas as alternativas propostas para mitigação de riscos em termos de seus custos, benefícios, riscos e da avaliação do impacto das decisões atuais sobre as alternativas futuras.

Um resumo dos processos no gerenciamento de riscos é definido a seguir. Seis são os tópicos incluídos nos grupos de atividades de planejamento, monitoramento e controle.

\begin{itemize}

\item Planejar o gerenciamento dos riscos: o processo de definição da condução das atividades de gerenciamento de risco num projeto;

\item Identificar riscos: o processo de determinação dos riscos que possam afetar o projeto e da documentação de suas características;

\item Realizar a análise qualitativa dos riscos: o processo de priorização dos riscos para análise ou ações adicionais através da avaliação e combinação de suas probabilidades de ocorrência e impacto;

\item Realizar a análise quantitativa dos riscos: o processo de análise numérica do efeito de riscos identificados previamente, em termos dos objetivos gerais do projeto;

\item Planejar respostas aos riscos: o processo de desenvolvimento de opções e ações para aumento das oportunidades e diminuição das ameaças aos objetivos do projeto;

\item Monitorar e controlar os riscos: o processo de implementação do planejamento de respostas aos riscos, rastreamento de riscos identificados, monitoramento dos riscos residuais, identificação de novos riscos e avaliação da eficácia do processo de tratamento de risco durante todo o projeto.

\end{itemize}

O Software Engineering Institute(SEI) desenvolveu uma metodologia de gerenciamento de risco chamada \textit{Software Risk Evaluation}(SRE) que é especificamente voltada para projetos na indústria de software. O paradigma SRE é composto por seis elementos: cinco módulos (identificação, análise, planejamento, acompanhamento e controle) e um elemento central (comunicação), que é a chave para a efetiva gestão de riscos \cite{HIGUERAHAIMES1996} \cite{williams1999software}.

Boehm \cite{BOEHM1991}, Chapman \cite{chapman1996project}, Fairley \cite{fairley1994risk}, Bandyopadhyay et al. \cite{bandyopadhyay1999framework} apresentaram métodos e modelos diferentes de gerenciamento de riscos em projetos de software. No entanto, há elementos em comum em todas as abordagens anteriores: a identificação, a avaliação da probabilidade e impacto, e o planejamento a respostas para manuseio desses riscos se eles ocorrerem \cite{holzmann2011developing}.

~~Para projetos,~~ O gerenciamento de risco, no sentido amplo, é útil para organizações em que muitos projetos são realizados. Mas a partir da perspectiva de um líder de um único projeto, há apenas aquele projeto. O gerenciamento de risco do ponto de vista da empresa, ou para um \textit{portfolio} de projetos, foca principalmente no risco agregado. Todavia, o genrenciamento de risco de projetos deve focar no risco pensando principalmente no sentido restrito \cite{kendrick2003identifying}. ?????? (NÃO ENTENDI O SENTIDO DO PARÁGRAFO. DESCREVA MELJHOR!!)

O gerenciamento de risco de projetos, no sentido restrito, serve para melhorar as chances de cada projeto individual alcançar o sucesso. Na maioria dos outros campos,(QUE CAMPOS ???) o gerenciamento de risco está preocupado principalmente com os valores médios de um grande número de eventos independentes. Para o gerenciamento de risco de projetos, no entanto, o que geralmente mais importa é a previsibilidade - gerenciar a variação esperada no resultado para o projeto específico \cite{kendrick2003identifying}.

Os objetivos da gestão de risco para um único projeto consistem no estabelecimento de ~~são estabelecer~~ um plano crível, credível consistente com os objetivos de negócio, e , em seguida, na minimização do intervalo de resultados possíveis. Quanto maior for o intervalo de duração possível para um projeto mais elevado será o seu risco. O risco do projeto aumenta com o nível de incerteza, tanto negativa quanto positiva \cite{kendrick2003identifying}.

O gerenciamento do risco de projetos utiliza dois parâmetros fundamentais de risco: probabilidade e perda. Probabilidade é geralmente a possibilidade de um evento ocorrer - como é frequentemente obtida através de suposição, logo pode ser bastante imprecisa. Perda é geralmente designado em projetos como "impacto", e baseia-se nas consequências para o projeto no caso de ocorrência do risco. Impacto é geralmente medido em tempo ou custo, particularmente para avaliação quantitativa de riscos \cite{kendrick2003identifying}.

O principal benefício do gerenciamento de riscos em projetos é tanto o desenvolvimento de uma base de credibilidade para cada projeto, mostrando que o mesmo isso é possível, quanto ou da demonstração de inviabilidade do projeto, mostrando que o mesmo deva podendo então ser evitado, abortado ou transformado. A análise de risco pode também revelar oportunidades para melhoria dos projetos que podem resultar em altos retornos no investimento. A análise de riscos adequada reduz tanto o custo total quanto a frustração causada por problemas evitáveis. A quantidade de retrabalho e atrasos imprevistos de esforço no projeto é reduzida. Além disso, a análise de risco revela fraquezas num plano de projeto e desencadeia mudanças, novas atividades e realocação de recursos que melhoram o resultado do projeto \cite{kendrick2003identifying}.

## Qualitative Risk Analysis

O processo de análise qualitativa avalia as características dos riscos de projetos identificados individualmente e prioriza-os baseado nas requisições estabelecidas para o projeto. Em outras palavras, a análise qualitativa avalia a probabilidade de cada evento ocorrer e o efeito individual de cada um deles nos objetivos do projeto. Como tal processo não aborda diretamente o risco global para os objetivos do projeto, que resultam do efeito combinado dos eventos individuais e as interações entre si, então isso pode ser obtido através do uso de técnicas de análise quantitativa \cite{PRACTICESTANDARD2009}.

## Quantitative Risk Analysis

Análise é a conversão de dados de risco em informação para tomada de decisão. A análise fornece a base para o gerente de projetos trabalhar nos riscos certos e mais críticos. Boehm \cite{BOEHM1991} define o objetivo da análise de risco como a avaliação da probabilidade e magnitude de perda para cada item de risco identificado, e ele avalia os riscos compostos das interações dos itens de risco. Técnicas típicas incluem modelos de desempenho e custo, análise de rede, análise de decisão estatística e análise de fatores de qualidade (tais como confiabilidade, disponibilidade e segurança).

A análise de risco depende de um bom mecanismo de identificação de riscos. No entanto, a maioria dos métodos assume que os gerentes devam ter a experiência necessária a respeito de todos os fatores de risco pertinentes, o que nem sempre corresponde a realidade. Além disso, muitos desses métodos podem ser demorados e, portanto, muito dispendiosos para se utilizar de forma regular. Portanto, um método popular para a identificação de fatores de risco tem sido a utilização de listas de verificação. Infelizmente, essas listas de verificação são baseadas em pequenas amostras ou, ainda pior, viciadas pelos seus métodos de coleta de dados históricos de risco.

As técnicas mais utilizadas para análise de risco incluem \cite{PMBOK2008}:

\begin{itemize}

\item Análise de Sensibilidade: ajuda a determinar quais riscos têm os maiores impactos potenciais no projeto. Uma representação típica disso é o diagrama de tornado??? (O QUE É ISTO??);

\item Valor Monetário Agregado(EMV): é um conceito estatístico que calcula o valor médio DE QUÊ??? quando o futuro inclui cenários que podem ou não ocorrer (ou seja, sob a análise da incerteza). Um uso comum desse tipo de técnica é a análise da árvore de decisão;

\item Modelagem e simulação: simulação utiliza um modelo que converte as incertezas especificadas e detalhadas em seu potencial impacto sobre os objetivos do projeto. As simulações iterativas gerais são realizadas usando Simulação de Monte Carlo;

\item Opinião especializada: opinião especializada (idealmente por especialistas com experiência relevante e recente) é necessária não apenas para identificação dos impactos potenciais sobre o custo e o cronograma e para avaliação da probabilidade, como também para definir as variáveis de entrada e quais ferramentas utilizar.

\end{itemize}

O objetivo da análise quantitativa de riscos é criar um "perfil do risco" do projeto. Para tanto, são necessárias as seguintes informações: a chance de o projeto ser finalizado dentro de um certo período de tempo ou orçamento; a taxa de sucesso de projetos; as estimativas de duração de pior caso, médio e melhor caso e outros parâmetros do projeto \cite{PMBOK2008}.

Alguns trabalhos propõem novas ferramentas de análise quantitativa de riscos. Entre eles, Virine \cite{VIRINE2009} apresenta a metodologia da Cadeia de Eventos. Nesse trabalho, as atividades de um projeto não são um procedimento uniforme e contínuo. Essas tarefas são afetadas por eventos externos, que transformam as atividades de um evento para outro. O momento em que os eventos externos ocorrem são probabilísticos e podem ser definidos utilizando uma distribuição estatística. Além disso, eventos podem causar outros eventos, criando, portanto, a Cadeia de Eventos. A análise dessas combinações é realizada através da Simulação de Monte Carlo.

A análise de risco demonstra a incerteza dos resultados do projeto e é útil para justificar reservas de cronograma e/ou recursos. É mais apropriado para definir uma janela de tempo (ou orçamento) em vez de um objetivo único para projetos arriscados. Por exemplo, o cronograma alvo para um projeto arriscado pode ser doze meses, mas o cronograma empenhado, refletindo problemas potenciais, pode ser fixado em quatorze meses. A conclusão no prazo (ou antes) desse intervalo define um projeto de sucesso; somente se o projeto demorar mais de quatorze meses será considerado um fracasso \cite{kendrick2003identifying}.

**Conventional Techniques for Risk Analysis**

**Monte Carlo Simulation**

A Simulação de Monte Carlo é uma técnica que computa ou itera o custo ou cronograma do projeto muitas vezes usando valores de entrada selecionados aleatoriamente a partir de distribuições de probabilidades de custos ou durações possíveis, para calcular uma distribuição dos custos totais ou datas de finalização do projeto \cite{PMBOK2008}.

Um modelo é desenvolvido, e ele contém algumas variáveis de entrada. Essas variáveis de entrada tem resultados possíveis diferentes, representados por uma função de distribuição de probabilidade de valores para cada variável. O método de Monte Carlo é uma abordagem de simulação através de computação intensiva para determinar a probabilidade de resultados possíveis de um objetivo do projeto, tais como a data de finalização ou o custo total. As entradas do procedimento são obtidas aleatoriamente a partir de intervalos específicos com funções de distribuição de probabilidade para as durações das atividades no cronograma ou itens da linha de base de custo. Esses diferentes valores de entrada são utilizados não apenas para construir um histograma de resultados possíveis para o projeto e sua probabilidade relativa, mas também a probabilidade cumulativa para calcular as reservas de contingências desejadas para o cronograma ou custo. Resultados adicionais incluem a importância relativa de cada entrada para determinar o custo e o cronograma geral para o projeto \cite{kwak2007exploring}.

O desenvolvimento A invenção desse método, especialmente o uso de computadores para realização dos cálculos, foi creditado a Stanislaw Ulam, um matemático que trabalhou no Projeto Americano de Manhattan durante a Segunda Guerra Mundial. O seu trabalho com Jon von Neuman e Nicholas Metropolis transformaram a amostragem estatística de uma curiosidade matemática para uma metodologia formal aplicável a uma grande variedade de problemas \cite{kwak2007exploring}.

A Simulação de Monte Carlo é uma abordagem detalhada de simulação através de computação intensiva para determinar a probabilidade de resultados possíveis de um objetivo do projeto; por exemplo, a data de conclusão ou o custo total. As entradas para o procedimento são obtidas aleatoriamente a partir de intervalos específicos com funções de distribuição de probabilidade para as durações das atividades do cronograma ou itens da linha de custo. Esses valores de entrada diferentes são usados para construir um histograma de possíveis resultados do projeto e sua probabilidade relativa, como também, a probabilidade cumulativa para calcular as reservas de contingência desejadas de tempo ou custo. Resultados adicionais incluem a importância relativa de cada entrada na determinação do custo geral do projeto e cronograma. Um exemplo de resultados de estimativa de riscos de cronograma e custo são apresentados na Figura \ref{fig:montecarlo} \cite{PRACTICESTANDARD2009}.

Na Figura \ref{fig:montecarlo}, observa-se a previsão de finalização para um cronograma de um projeto. No eixo horizontal, encontram-se as possíveis datas de finalização do cronograma, a frequência de ocorrência dessas datas na amostragem são representadas pelas barras verticais, a frequência cumulativa é representada pela curva escura e pelos respectivos valores na borda direita do gráfico. A partir da frequência relativa, é possível prever a probabilidade do projeto finalizar até determinada data, nesse exemplo, esse projeto teve 90\% de chance de finalizar até 08 de Maio de 2009.

**PERT Analysis**

FALTA PREENCHER ESSA SEÇÃO!! ?????????????????????????????????????

## Statistical and Intelligent Computing Techniques for Risk Analysis

Nesta seção, serão explorados inicialmente dois modelos de previsão que poderiam ser aplicados para o domínio de análise de risco: regressão linear múltipla e modelos de árvore de regressão. A escolha não foi conseqüência de alguma etapa de seleção formal de modelos, contudo, ainda assim, os modelos apresentam-se como duas boas alternativas para problemas de regressão como são bastante diferentes em termos de suas suposições sobre a "forma" da função de regressão sendo aproximada e pela facilidade de interpretar e velocidade para rodar em qualquer computador. O objetivo da escolha desses dois modelos de regressão comuns está relacionado com a validação do desempenho de outras técnicas. Por exemplo, se uma técnica de previsão tem um desempenho pior do que modelos de regressão múltipla linear, como de árvores de regressão, então parece ser uma alternativa inadequada ruim.

Multiple Linear Regression

A Regressão Linear Múltipla está entre as técnicas de análise de dados estatísticos mais utilizadas. Esse modelo obtém uma função aditiva relacionando uma variável de objetivo a um conjunto de variáveis de previsão. Essa função aditiva é a soma de elementos da fórmula $\beta\_i \times X\_i$, onde $X\_i$ é uma matriz estimadora de variáveis e $\beta\_i$ é a matriz de pesos na regressão linear múltipla \cite{torgo2003data}

Na regressão linear, os dados são modelados para caber numa linha reta. Por exemplo, uma variável aleatória $Y$, chamada variável de resposta, pode ser modelada como uma função linear de uma variável aleatória $X$, chamada de variável de previsão com a Equação \ref{eq:mlr}:

\begin{equation}

\label{eq:mlr}

Y = \alpha + \beta X,

\end{equation}

onde a variância de $Y$ é assumida ser constante. Os coeficientes $\alpha$ e $\beta$ (chamados coeficientes de regressão) especificam a interceptação de $Y$ e a inclinação da linha, respectivamente. Esses coeficientes podem ser resolvidos pelo método dos mínimos quadrados, que minimiza o erro entre a linha real separando os dados e a estimativa da linha. Dado $s$ amostras ou pontos de dados da forma $(x\_1,y\_1), (x\_2,y\_2), ..., (x\_s,y\_s)$, então os coeficientes de regressão podem ser estimados usando esse método por meio das Equações \ref{eq:betamlr} and \ref{eq:alphamlr}:

\begin{equation}

\label{eq:betamlr}

\beta = \frac{\sum\limits\_{i=1}^{s} (x\_i - x) (y\_i - y) }{\sum\limits\_{i=1}^{s} (x\_i - x)^2},

\end{equation}

\begin{equation}

\label{eq:alphamlr}

\alpha = y - \beta x,

\end{equation}

onde $x$ é a média de $x\_1,x\_2,...,x\_s$, e $y$ a média de $y\_1,y\_2,...,y\_s$. Os coeficientes $\alpha$ e $\beta$ frequentemente provêm boas aproximações mesmo para equações de regressão complexas. A regressão múltipla é uma extensão da regressão linear, permitindo uma resposta variável $Y$ para ser modelada como uma função linear de um vetor de características multidimensional \cite{han2006data}.

**Regression Tree Model**

Uma árvore de regressão é uma hierarquia de testes lógicos em algumas das variáveis explanatórias do modelo. Logo, nem todas as variáveis aparecem na árvore. Uma árvore é lida do nó-raiz para os nós-folha. Cada nó da árvore tem dois galhos. Eles estão relacionados ao valor de um teste em cada uma das variáveis de previsão. O teste continua até um nó-folha ser alcançado. Nesses nós-folha tem-se os valores previstos pela árvore. Isso significa que se é necessário usar uma árvore para obter-se uma estimativa para uma amostra particular, é necessário seguir um galho do nó-raiz até um nó-folha, de acordo com os resultados do teste para a amostra. O valor médio da variável alvo encontrado no nó-folha alcançado é a previsão da árvore \cite{torgo2003data} \cite{breiman1984classification}.

\begin{figure}[h]

\centering

\includegraphics[width=\columnwidth]{image/regressiontreemodel.pdf}

\caption{Regression Tree Model for PERIL}

\label{fig:rtm}

\end{figure}

A Figura \ref{fig:rtm} apresenta o modelo de árvore de regressão para a base de dados do PERIL. Partindo do nó-raiz é possível verificar dois galhos que mostram testes lógicos para a variável \textit{Date0}. Os próximos nós representam resultados possíveis levando-se em conta somente \textit{Date0}. O segundo nível da árvore mostra testes lógicos associados às variáveis \textit{Category3} e \textit{Subcat2}. Em seguida, encontram-se três nós-folha. Cada valor do nó-folha representa os possíveis resultados e o número de amostras no PERIL, que satisfazem os testes lógicos conjugados nos níveis anteriores da árvore. Por último, testes lógicos para a variável \textit{Subcat0} são apresentados para geração de outros dois nós-folha.

Árvores são obtidas geralmente em dois passos. Inicialmente, uma grande árvore é desenvolvida, e em seguida essa árvore é podada eliminando os nós inferiores através de um processo de previsão estatística. Esse processo tem o objetivo de evitar o \textit{overfitting}. Isso tem a ver com o fato de que uma árvore larga demais irá memorizar (~~decorar~~) os dados do treinamento de uma maneira quase perfeita, mas irá capturar relações espúrias do conjunto de dados oferecido (\textit{overfitting}), e portanto irá ter um desempenho ruim quando exposta a uma nova amostra de dados para a qual previsões são necessárias. O problema de \textit{overfitting} ocorre em muitas técnicas de modelagem, particularmente quando as suposições acerca da função a ser aproximada têm menor importância. Esses modelos, mesmo que tenham uma ampla faixa de aplicação (devido a essas condições não prioritárias), sofrem com esse problema de \textit{overfitting} necessitando, portanto, de uma previsão baseada em estatística para diminuir o impacto desse efeito \cite{torgo2003data}.

**Artificial Neural Networks**

Uma Rede Neural Artificial (RNA) é um processador distribuído maciçamente paralelo constituído por unidades de processamento simples, que tem a propensão natural para armazenar conhecimento experimental e torná-lo disponível para o uso \cite{linlee1996neuralfuzzy}. Ela adota estimativas de regressão não-paramétricas constituída de elementos de processamento interconectados entre dados de entrada e saída, e tem uma excelente capacidade de aprendizado e generalização.

Outra definição descreve uma RNA como um sistema constituído de elementos de processamento, chamados neurônios, os quais estão dispostos em camadas (uma camada de entrada, uma ou várias camadas intermediárias e uma camada de saída) e são responsáveis pela não-linearidade e pela memória da rede \cite{valenca2005aplicando}.

As RNA Redes Neurais Artificiais (RNA) são modelos que procuram simular o comportamento e o funcionamento do cérebro humano. Assim como existem neurônios biológicos, componentes essenciais para o processamento das informações do cérebro, a RNA é formada por unidades que objetivam realizar as mesmas funções do neurônio biológico. Esses componentes são denominados neurônios artificiais e foram propostos em 1943 por Mc-Culloch e Pitts \cite{MCCULLOCKPITTS1943}.

Em seguida ao trabalho de McCulloch e Pitts surgiu a regra de aprendizagem proposta por Donald Hebb \cite{hebb1949organization} que se constitui a base de todas as regras de aprendizagem. Em seu famoso livro, o autor procurou encontrar um mecanismo neural capaz de explicar como as informações podiam ser armazenadas e recuperadas dos neurônios. A regra de aprendizagem foi enunciada da seguinte forma: "Quando um neurônio recebe um estímulo de outro neurônio, e se ambos estão altamente ativos, o peso entre eles deve ser fortalecido, caso contrário enfraquecido". Em 1960, Widrow e Hoff \cite{widrow1960adaptive} apresentaram uma regra de aprendizagem para uma extensão do Perceptron (CARLOS, DESCREVA ANTES DE FALAR SOBRE O PERCEPTRON, QUEM ESTIVER LENDO PRECISA SABER DO QUE SE TRATA), desenvolvida previamente por Frank Rosenblatt \cite{rosenblatt1960perceptron}, chamada de ADALINE (Adaptive Linear Neuron). Esta regra baseada no método dos mínimos quadrados ficou conhecida como regra delta. Paul Werbos em 1974 \cite{Werbos74} desenvolveu o algoritmo de backpropagation, sendo esse algoritmo posteriormente popularizado através da publicação feita por Rumelhart e McClelland em 1985 \cite{rumelhart1985learning}.

O modelo do neurônio de McCulloch e Pitts procura representar o neurônio biológico utilizando uma regra de propagação e uma função de ativação. Considere $x\_1, x\_2, x\_3, ..., x\_n$, como sendo as variáveis de entrada $x\_j$, em que $j = 1,...,n$ do neurônio de saída $i$. A entrada líquida $net\_i$ é dada pela seguinte regra de propagação:

\begin{equation}

\label{eq:net}

net\_i = \sum\_{j=1}^{n} w\_{ij}x\_j - \theta

\end{equation}

onde, $w\_{ij}$ são os pesos sinápticos e $\theta$ é o limiar de ativação do neurônio.

\begin{figure}[h]

\centering

\includegraphics[width=.6\textwidth]{image/neuronio.png}

\caption{Representação gráfica da MLP com três camadas}

\label{fig:neuronio}

\end{figure}

A saída $y\_i$ é dada por $f(net\_i)$, em que $f$ é a função de ativação. Existem várias funções de ativação propostas, dentre elas as mais comuns são: linear, degrau, rampa, sigmoide logística, tangente hiperbólica e gaussiana representadas respectivamente pelas equações seguintes \cite{valenca2005aplicando} \cite{engelbrecht2007computational}:

\begin{equation}

\label{eq:linear\_activation}

f(net\_i) = net\_i

\end{equation}

\begin{equation}

\label{eq:step\_activation}

f(net\_i) =

\begin{cases}

\gamma\_1 & \text{if } net\_i \geq \theta \\

\gamma\_2 & \text{if } net\_i < \theta

\end{cases}

\end{equation}

\begin{equation}

\label{eq:ramp\_activation}

f(net\_i) =

\begin{cases}

\gamma & \text{if } net\_i \geq \epsilon \\

net\_i & \text{if } -\epsilon < net\_i < \epsilon \\

-\gamma & \text{if } net\_i \leq -\epsilon

\end{cases}

\end{equation}

\begin{equation}

\label{eq:logisticsigmoid\_activation}

f(net\_i) = \frac{1}{1 + e^{-net\_i}}

\end{equation}

\begin{equation}

\label{eq:hiberbolictangent\_activation}

f(net\_i) = \frac{e^{net\_i} - e^{-net\_i}}{e^{net\_i} + e^{-net\_i}}

\end{equation}

\begin{equation}

\label{eq:gaussian\_activation}

f(net\_i) = e^{-(net\_i - \theta)^2/\sigma^2}

\end{equation}

**MultiLayer Perceptron**

A rede perceptron de múltiplas camadas é uma generalização da rede perceptron simples pela adição de pelo menos uma camada intermediária, conhecida como camada escondida. Em uma rede em camadas, os neurônios estão dispostos em cada uma delas. Na MLP, a primeira delas é a camada de entrada, na qual as variáveis de entrada são conectadas diretamente a neurônios exclusivos ???? ~~a um neurônio exclusivo~~ ???. A próxima camada, denominada intermediária, liga completamente os neurônios da camada anterior aos neurônios da camada de saída. Por fim, a camada de saída representa a saída da RNA. Cada neurônio de entrada ~~em um neurônio~~ tem um peso a ele associado a ser ajustado pelo algoritmo de treinamento. Um modelo comum de MLP contém um neurônio de bias(EXPLICAR). A MLP é um grafo direto, no qual as entradas de dados são propagadas a partir da camada de entrada para a\(s\) camada\(s\) escondida\(s\) e da\(s\) camada\(s\) escondida\(s\) para a camada de saída. O fluxo de dados no caminho para frente numa MLP é conhecido como "fase forward". O fluxo de dados na direção oposta é denominado de "fase backward".

Uma das principais preocupações da ANN é o dilema da estabilidade-plasticidade (EXPLICAR O QUE É ANN E O DILEMA CITADO). Embora a aprendizagem contínua seja desejada numa RNA, a aprendizagem futura fará com que a RNA perca sua memória quando os pesos alcançaram um estado de equilíbrio \cite{haykin1994neural}. O algoritmo \textit{Backpropagation}, a ser definido mais adiante, é comumente usado como o método de treinamento, pois nos permite ajustar os pesos da rede de múltiplas camadas, através da Regra Delta Generalizada \cite{rumelhart1985learning}.

A vantagem de ter camadas intermediárias é que a rede neural passa a resolver problemas que não são linearmente separáveis, possibilitando, assim, a aproximação de qualquer função contínua, com apenas uma camada, e qualquer função matemática, quando houver mais de uma camada \cite{HAYKIN2007}. A Figura \ref{fig:mlp} ilustra a forma gráfica da MLP, apresentando as entradas, saídas e as camadas de entrada, intermediária e de saída.

\begin{figure}[h]

\centering

\includegraphics[width=.6\textwidth]{image/mlp.png}

\caption{Representação gráfica da MLP com três camadas}

\label{fig:mlp}

\end{figure}

\begin{figure}[h]

\centering

\includegraphics[width=.6\textwidth]{image/mlp2.png}

\caption{Representação gráfica da MLP com três camadas}

\label{fig:mlp2}

\end{figure}

O funcionamento da MLP é dividido em duas fases: \textit{forward} e \textit{backward}. Na fase \textit{forward}, um neurônio de uma camada está ligado a todos os neurônios da camada seguinte. Os sinais da entrada são propagados da camada de entrada para a camada escondida e da camada escondida para a camada de saída; cada neurônio processa as entradas e apresenta uma saída. Nessa fase é possível calcular o erro entre a saída desejada para a rede e a saída apresentada pela MLP. Na fase \textit{backward} o erro é retropropagado e os pesos são ajustados, utilizando o algoritmo de ajustes de pesos, inicialmente aleatórios, chamado \textit{Backpropagation} \cite{valenca2005aplicando}.

**Backpropagation Algorithm**

A primeira etapa do Backpropagation consiste na inicialização dos pesos com valores aleatórios. A partir da inicialização, para cada exemplo da base de treinamento é realizado o seguinte processo: a propagação do sinal, a fase \textit{forward} e retropropagação do erro, a fase \textit{backward}. Na fase \textit{forward} é possível calcular o erro entre a saída desejada para a rede e a saída apresentada pela MLP. Na fase \textit{backward} o erro é retropropagado e os pesos são ajustados. Faz-se necessário calcular a sensibilidade para cada neurônio. A sensibilidade para as camadas é dada por:

\begin{equation}

\label{eq:sensibiliy\_function}

\delta^{m-1}\_i = f^{m-1'}(net^{m-1}\_j)\sum\_{i=1}^{N\_{neurons}}w^m\_{ij}\delta^m\_i

\end{equation}

em que, $w^m\_{ij}$ são os pesos sinápticos que serão ajustados, $\delta^m\_i$ representa a sensibilidade, o índice $i$ representa o número de neurônios da camada que recebe o sinal e possui $N\_{neurons}$ e $f^{m-1'}(net^{m-1}\_j)$ é a derivada da função de ativação dos neurônios da camada "M-1" (camada que emite o sinal) em relação à entrada líquida, $net^{m-1}\_j$.

O ajuste dos pesos é dado por:

\begin{equation}

\label{eq:backpropagation\_function}

w^m\_{ij}(t+1) = w^m\_{ij}(t) + \alpha \delta^m\_i y^{m-1}\_j + \beta \Delta w^m\_{ij}(t-1)

\end{equation}

em que $\alpha$ é a taxa de aprendizagem do algoritmo, $\beta$ é a taxa de momento e $\Delta w^m\_{ij}(t-1)$ é a correção realizada no peso $w\_{ij}^m$ na iteração $t-1$. Quanto maior o valor de $\alpha$ maiores serão as correções realizadas a cada iteração. Já o momento $\beta$ tem o objetivo de deixar o algoritmo menos susceptível a ficar preso em mínimos locais (no algoritmo original,$\beta = 0$), devido a superfície da função erro ser bastante complexa.

Durante o treinamento os exemplos vão sendo apresentados à MLP de forma aleatória e os pesos vão sendo ajustados pelo algoritmo \textit{backprogation} até que se chegue a uma condição de parada. A parada do treinamento é uma decisão crítica, pois uma parada prematura do treinamento poderá acarretar um \textit{underfitting} (os pesos da MLP não são ajustados de forma satisfatória), e caso o treinamento se torne excessivamente longo, pode ocorrer o \textit{overfitting} (a rede memoriza os exemplos de treinamento e ocorre perda na capacidade de generalização) \cite{valenca1995fundamentos}. Objetivando evitar essas duas condições, e consequentemente uma melhor capacidade de generalização, a base de dados para treinamento da rede pode ser dividida em duas partes: uma parte de fato para treinamento e uma outra parte para validação. A parte para treinamento é apresentada à rede e serve para realizar o ajuste dos pesos pelo algoritmo de treinamento. A parte correspondente ao conjunto de validação serve para realizar uma avaliação do treinamento da rede. Quando o erro médio quadrático para o conjunto de validação começar a aumentar, significa que a MLP está memorizando o conjunto de treinamento e neste momento é adequado interromper o treinamento.

As redes MLP são uma boa ferramenta para a construção de classificadores, já que além de conseguir lidar com o mapeamento entrada-saída não linear, apresentam alto poder de generalização \cite{haykin-1994}.

Em \cite{HUHUANG2007}, algumas técnicas inteligentes são utilizadas para a análise do risco de projetos de software. O artigo conclui que uma técnica híbrida de redes neurais e algoritmos genéticos obtem uma precisão de 85\% na predição do projeto obter sucesso, ser desafiado ou falhar totalmente.

**Support Vector Machine**

As Máquinas de Vetores de Suporte (\textit{Support Vector Machine}, ou SVMs) são redes neurais artificiais que foram desenvolvidas tendo por base a teoria do aprendizado estatístico, detalhada por Vapnik em 1995 e 1998 \cite{vapnikSnature} \cite{vapnik1998statistical} e inicialmente proposta por Vapnik e Chervonenkis em 1971 \cite{vapnik1971uniform}.

O desenvolvimento das redes SVMs teve por objetivo obter redes neurais com alta capacidade de generalização, uma vez que, durante o treinamento destas redes se busca não apenas minimizar o erro de treinamento mas também a complexidade da rede obtida. De acordo com a teoria do aprendizado estatístico este erro de generalização chamado de Risco Funcional pode ser calculado caso se conheça a distribuição de probabilidade da população \cite{vapnik1998statistical}. Entretanto, para problemas de aprendizado supervisionado, em geral não se dispõe da distribuição de probabilidade da população,uma vez que em problemas reais o que se utiliza é uma amostra da população sendo possível apenas o cálculo de uma função erro, como por exemplo o erro médio quadrático. Esta função erro calculada da amostra de dados é chamada de Risco Empírico. Logo, a depender do tamanho da amostra utilizada para treinamento, a minimização do erro de treinamento, não significa necessariamente uma minimização do erro de generalização ou Risco Funcional.

Um conceito importante da teoria do aprendizado estatístico é o da dimensão VC (Vapnik-Chervonenkis). A dimensão VC é um índice escalar que mede a complexidade intrínseca de uma classe de funções. Uma definição para a dimensão VC é de que esta representa o número máximo de exemplos de treinamento que uma máquina de aprendizagem é capaz de classificar corretamente, para todas as possíveis combinações binárias desses dados.

Desta forma, de acordo com a teoria do aprendizado estatístico proposta por Vapnik \cite{vapnik1998statistical}, uma rede neural ou qualquer outra máquina de aprendizagem durante o aprendizado supervisionado obtém sua capacidade máxima de generalização quando durante o treinamento se minimiza o Risco Funcional. A minimização do Risco Funcional é equivalente a minimização do Risco Empírico e do Risco Estrutural associado a complexidade do modelo. O Risco Funcional é limitado, com probabilidade $1 - \delta$ pela Equação \ref{eq:svm\_functional\_risk}.

\begin{equation}

\label{eq:svm\_functional\_risk}

R\_{funcional} \leq R\_{empirico} + R\_{estrutural}

\end{equation}

O Risco Estrutural por sua vez pode ser calculado, conforme mostrado por Vapnik \cite{keylist} como:

\begin{equation}

\label{eq:svm\_structural\_risk}

R\_{estrutural} = \sqrt{\frac{h \left( \log \left( \frac{2N}{h} \right) + 1 \right) - \log \left( \frac{\delta}{4} \right)}{N}}

\end{equation}

onde $h$ é a dimensão VC.

A formulação inicial das SVMs para problemas linearmente separáveis foi chamada de SVMs de margens rígidas (máximas). A diferença destas para uma rede \textit{perceptron} está na forma de seleção do hiperplano de separação das classes. Enquanto numa rede \textit{perceptron} se busca qualquer hiperplano que satisfaça ao problema, numa SVM de margem rígida se procura o hiperplano ótimo, ou seja, aquele cuja margem de separação é máxima.

Desta forma, a determinação dos pesos das SVMs se tranforma em um problema de otimização com restrição, o que exige um maior esforço computacional. A otimização dos pesos das SVMs com restrição de margem máxima tem sido resolvida através do método de multiplicadores de Lagrange.

Posteriormente, foram elaboradas as SVMs de margens flexíveis (suaves) com o objetivo de lidar com exemplos de treinamento com erros ou ruídos e finalmente as SVMs não lineares que utilizam na camada escondida funções de base diferentes.

Para um melhor entendimento das SVMs considere um problema de classificação linear com vetor de entrada $x\_i$ $(i=1,...,n)$, onde $n$ é o número de exemplos e saídas $y\_i$(+1 ou -1). A seleção dos parâmetros deve ocorrer de tal forma que:

\begin{eqnarray}

\label{eq:svm\_example}

w^T.x\_i + b \geq +1 ,& se& y\_i= +1 \\

w^T.x\_i + b \leq -1 ,& se& y\_i= -1

\end{eqnarray}

ou de forma resumida

\begin{eqnarray}

\label{eq:svm\_example\_summary}

y\_i (w^T.x\_i + b) \geq 1 ,&& i=1,..,n

\end{eqnarray}

A margem (d) é a distância entre os dois planos paralelos e pode ser expressa como uma distância euclidiana, na Equação \ref{eq:svm\_euclian\_distance}.

\begin{equation}

\label{eq:svm\_euclian\_distance}

d = \frac{| w^T.x\_i + b |}{|| w ||} + \frac{| w^T.x\_i + b |}{|| w ||} = \frac{2}{|| w ||}

\end{equation}

onde, $||w|| = \sqrt{w\_1^2 + w\_2^2 + ... + w\_n^2}$ é a norma Euclidiana do vetor de pesos.

Portanto, maximizar a margem se transforma em um problema de otimização com restrição para minimizar a seguinte função objetivo $\min\_w \frac{||w||^2}{2}$ sujeito a Equação \ref{eq:svm\_example\_summary}.

Esse é um problema de otimização não-linear (a função objetivo é quadrática) com restrições lineares que pode ser resolvido por meio dos multiplicadores de Lagrange.

A Máquina de Vetor de Suporte, do inglês \textit{Support Vector Machine} (SVM), é uma técnica de aprendizado de máquina aplicável a problemas de reconhecimento de padrões nos quais se busca atingir alto potencial de generalização \cite{HAYKIN2007} \cite{valenca2005aplicando}. O objetivo da SVM é encontrar um hiperplano particular, denominado de hiperplano ótimo, que maximize a margem de separação, conforme pode ser visualizado na Figura \ref{fig:svm}. In Figure \ref{fig:svm}, observamos dois vetores de suporte capazes de separar linearmente as saídas no hiperplano em duas classes. A variável $r$ é a distância algébrica desejada dos vetores de suporte para o hiperplano ótimo de separação das classes. Quanto maior essa distância, maior a capacidade de generalização da máquina de vetor de suporte.

**Radial Basis Function Network**

As redes \textit{Radial Basis Function }(RBF) surgiram em 1988 como uma possível alternativa às redes MLP. Na sua formulação tradicional é composta por três camadas: uma camada de entrada, uma camada escondida e uma camada de saída. As principais diferenças entre as redes RBF e MLP são:

\begin{enumerate}

\item Os neurônios da camada intermediária têm apenas funções de base radial como função de ativação, que são funções localizadas de tal maneira que apenas algumas unidades escondidas ficarão ativadas ao receberem um dado conjunto de exemplos de entrada;

\item Nas redes MLP as funções de ativação têm como entrada líquida uma média ponderada entre os exemplos de entrada e o conjunto de pesos. Por outro lado, nas redes RBF a utilização destas funções de ativação de base radial fazem com que sua ativação seja obtida a partir de uma norma ponderada da diferença entre o valor de entrada e o centro da função de base radial;

\item A camada de saída é composta por unidades de processamento lineares.

\end{enumerate}

As funções de ativação mais utilizadas são \cite{valenca2005aplicando} \cite{engelbrecht2007computational}:

\begin{enumerate}

\item Função Linear

\begin{equation}

\label{eq:rbf\_linear\_function}

\phi\_i(x) = || x\_j - \mu\_i ||

\end{equation}

\item Função Cúbica

\begin{equation}

\label{eq:rbf\_cubic\_function}

\phi\_i(x) = || x\_j - \mu\_i ||^3

\end{equation}

\item Função de base Gaussiana

\begin{equation}

\label{eq:rbf\_gaussian\_function}

\phi\_i(x) = exp \left( - \frac{|| x\_j - \mu\_i ||^2}{2\sigma\_{i}^{2}} \right)

\end{equation}

\item Função Logística

\begin{equation}

\label{eq:rbf\_logistic\_function}

\phi\_i(x) = \frac{1}{1 + exp \left( - \frac{|| x\_j - \mu\_i ||^2}{\sigma\_{i}^{2} - \theta} \right)}

\end{equation}

\item Função de base Multi-quadrática

\begin{equation}

\label{eq:rbf\_multiquadratic\_function}

\phi\_i(x) = \sqrt{|| x\_j - \mu\_i ||^2 + \sigma\_i^2}

\end{equation}

\item Função de base Multi-quadrática inversa

\begin{equation}

\label{eq:rbf\_inverse\_multiquadratic\_function}

\phi\_i(x) = \frac{1}{\sqrt{|| x\_j - \mu\_i ||^2 + \sigma\_i^2}}

\end{equation}

\item Função de base lâmina Spline fina

\begin{equation}

\label{eq:rbf\_spline}

\phi\_i(x) = \frac{|| x\_j - \mu\_i ||}{\sigma\_i^2} \log \left( \frac{|| x\_j - \mu\_i ||}{\sigma\_i} \right)

\end{equation}

\end{enumerate}

onde, $x\_j$ são exemplos de entrada, $\mu\_i$ e $\sigma\_i$ representam respectivamente o centro e a largura (dispersão) da i-ésima função de base radial e $\theta$ é um bias ajustado.

O cálculo da resposta da rede RBF correspondente aos neurônios da camada de saída é realizado pela Equação \ref{eq:rbf\_output\_formula}.

\begin{equation}

\label{eq:rbf\_output\_formula}

y\_j = \sum\_{j=1}^{s} w\_{Kj} \phi\_K(x)

\end{equation}

O treinamento das redes RBF pode ser realizado de diversas formas. Dentre as abordagens propostas, o treinamento mais empregado é chamado de treinamento híbrido, por utilizar uma aprendizagem não supervisionada para determinar os parâmetros das funções de base radial da camada escondida e um aprendizado supervisionado para ajustar os pesos que ligam a camada escondida a de saída. Portanto, desde que as funções de base radial, após determinação de seus parâmetros, sejam consideradas fixas, o ajuste dos pesos que ligam a camada escondida para a camada de saída para a rede RBF fica equivalente a uma rede ADALINE. Neste caso, ao se utilizar neurônios lineares na camada de saída e uma função objetivo como o erro médio quadrático, o treinamento destas redes é bastante rápido, uma vez que dispomos de um sistema de equações lineares para ser resolvido, o que nos permite utilizar técnicas lineares de inversão de matriz.

Durante a primeira fase de treinamento, aprendizagem não supervisionada, os centros das funções de base radial podem ser determinados por algoritmos de clusterização, tais como algoritmos k-médias \cite{musavi1992training} e mapas auto-organizáveis de Kohonen \cite{valenca1995fundamentos}.

O método k-média tem por objetivo encontrar um conjunto de $K$ centros $\mu\_i$ (i=1,2,...,$K$) representativos das funções de base radial em função do conjunto de exemplos de entrada $x\_j$ com $j=1,2,...,N$. O algoritmo divide o conjunto de exemplos de entrada em K subconjuntos $S\_i$cada um deles contendo $N\_i$ exemplos. O objetivo do método é minimizar a Equação \ref{eq:rbf\_minimization\_function}.

\begin{equation}

\label{eq:rbf\_minimization\_function}

F = \sum\_{i=1}^{K} \sum\_{j \in S\_i} || x\_j-\mu\_i ||^2

\end{equation}

onde, $\mu\_i$ é a média dos pontos pertencentes ao conjunto $S\_i$ calculada por

\begin{equation}

\label{eq:rbf\_mi\_i}

\mu\_i = \frac{1}{N\_i} \sum\_{j \in S\_i} x\_j

\end{equation}

Inicialmente, os exemplos de entrada são localizados aleatoriamente dentro de cada subconjunto $K$ e então os centros $\mu\_i$ são calculados. Posteriormente, os exemplos são redistribuídos de acordo com a sua maior proximidade com relação a estes centros. Este processo é então repetido até que não ocorram mais mudanças de exemplos entre os grupos. Este método permite também determinar o valor da largura (dispersão) $\sigma\_i$ da função de base radial por meio da distância euclidiana.

Logo, após a finalização do treinamento não-supervisionado, onde foram determinados os parâmetros das funções de base radial, se realiza a segunda parte do treinamento que consiste em um treinamento supervisionado onde se determinam os pesos que ligam a camada escondida à camada de saída. Nessa fase, o treinamento é similar a uma rede tradicional com a utilização de uma função objetivo tal como o erro médio quadrático. Diversas técnicas de otimização podem ser utilizadas para o aprendizado dos pesos da camada de saída, responsáveis pela combinação linear das ativações da camada escondida, tais como: a regra delta, o método dos mínimos quadrados, a técnica de pseudo-inversão e o método OLS (\textit{Orthogonal Least Squares}) \cite{chen1991orthogonal}.

As redes RBF são aproximadores universais de funções tal como a rede MLP. Em geral as redes RBF têm um tempo de treinamento inferior ao de uma rede MLP, uma vez que o treinamento híbrido da RBF permite a utilização de técnicas lineares de rápida convergência para determinação dos pesos (entre as camadas escondida e a de saída) durante o aprendizado supervisionado.

As redes RBF por serem redes de aprendizado local, necessitam de uma quantidade maior de exemplos o seu treinamento, para que se possa obter uma precisão similar a das redes MLP, que são redes de ajuste global. Isto implica que as redes MLP em geral são melhores aproximadores de funções, pois o ajuste global tende a fornecer uma maior capacidade de generalização. Entretanto, em problemas de classificação, as redes MLP tendem a cometer maiores erros de classificação "falso positivo" do que as redes RBF \cite{valenca1995fundamentos}.

\subsection{Learning Rules}

Uma propriedade de importância primordial para uma rede neural é a sua habilidade de aprender a partir de seu ambiente e de melhorar o seu desempenho. A melhoria do desempenho ocorre com o tempo de acordo com alguma medida preestabelecida. Uma rede neural aprende acerca do seu ambiente através de um processo iterativo de ajuste de seus pesos sinápticos e níveis de bias. Idealmente, a rede se torna mais instruída sobre o seu ambiente após cada iteração do processo de aprendizagem.

\subsubsection{Gradient Descent Learning}

O Gradiente Descendente (GD) requer a definição de uma função erro (ou objetivo) para medir o erro do neurônio na aproximação do alvo. A soma dos erros quadrados é normalmente usada \ref{eq:sse}. O objetivo do GD é encontrar os valores de peso que minimizem a função de erro. Isto é obtido através do cálculo da inclinação da função de erro no espaço de pesos, e para mover o vetor de peso ao longo do gradiente negativo, tal como ilustrado por um único erro na Figura \ref{fig:gd} \cite{engelbrecht2007computational}.

\begin{equation}

\label{eq:sse}

\epsilon = \sum\limits\_{i=1}^{P\_T}(e\_i - c\_i)^2

\end{equation}

onde $e\_i$ e $c\_i$ são respectivamente a saída esperada e calculada para o i-ésimo padrão, e $P\_T$ é o número total de pares de vetores entrada-saída no conjunto de treinamento.

\begin{figure}[h]

\centering

\includegraphics[width=.8\textwidth]{image/gd.png}

\caption{Vetores de Suporte e Hiperplano de Separação ótimo}

\label{fig:gd}

\end{figure}

Dado um padrão único de treinamento, os pesos são atualizados usando:

\begin{equation}

\label{eq:gd1}

v\_i(t) = v\_i(t-1) + \Delta v\_i(t)

\end{equation}

onde

\begin{equation}

\label{eq:gd2}

\Delta v\_i(t) = \eta (- \frac{\partial \epsilon}{\partial v\_i})

\end{equation}

e $\eta$ é a taxa de aprendizagem (isto é, o tamanho dos passos dados na direção contrária ao gradiente). A regra de aprendizagem Widrow-Hoff apresenta uma solução para as funções de passo e rampa, enquanto a regra de aprendizagem delta generalizada assume funções contínuas que sejam pelo menos diferenciáveis uma vez \cite{engelbrecht2007computational}.

\subsubsection{Widrow-Hoff Learning}

A regra de aprendizagem Widrow-Hoff assume que se a função objetivo $f = net\_i$, então $\frac{\partial f}{\partial net\_i}$. Essa regra de aprendizagem também é conhecida como o algoritmo \textit{least-mean-square} (LMS), foi um dos primeiros algoritmos usados ​​para treinar redes neurais em camadas com vários neurônios lineares adaptativos (Madaline) \cite{engelbrecht2007computational}.

**Error Correction Learning**

A regra delta é uma generalização da regra de Widrow-Hoff \cite{widrow1960adaptive} que utiliza funções de ativação que tenham derivada real em todo seu domínio e ao menos um valor mínimo. A regra de aprendizagem é dada por:

\begin{equation}

\label{eq:deltarule}

\Delta w\_{ij} = \alpha(d\_i - y\_i)x\_jf'(net\_i)

\end{equation}

ou seja, isso significa que o ajuste feito em um peso sináptico de um neurônio é proporcional ao produto do sinal de erro pelo sinal de entrada da sinapse em questão \cite{haykin-1994}. Assim, o ajuste a ser aplicado aos pesos é:

\begin{equation}

\label{eq:peso\_novo}

w\_{ij}(new) = w\_{ij}(old) + \alpha(d\_i - y\_i)x\_jf'(net\_i)

\end{equation}

Esse algoritmo garante a minimização do erro ao longo do tempo, através do ajuste dos pesos, ou seja, sua convergência garante que a adaptação dos pesos seja realizada num número finito de iterações. A prova de convergência pode ser encontrada em Beale e Jackson \cite{beale2010neural}.

**Memory based Learning**

Na aprendizagem baseada em memória, todas as (ou a maioria das) experiências passadas são armazenadas explicitamente em uma grande memória de exemplos de entrada-saída classificados corretamente: $ \{ ( x\_i, d\_i ) \}^{N}\_{i=1}$, onde $x\_i$ representa um vetor de entrada e $d\_i$ representa a resposta desejada correspondente. Sem perda de generalidade, pode-se restringir a resposta desejada a um escalar. Em um problema de classificação de padrões binário, por exemplo, há duas classes/hipóteses a serem consideradas, representadas por $\zeta\_1$ e $\zeta\_2$. Neste exemplo, a resposta desejada $d\_i$ assume o valor 0(ou -1) para a classe $\zeta\_1$ e o valor 1 para a classe $\zeta\_2$. Quando se deseja classificar um vetor de teste $x\_teste$ (não visto antes), o algoritmo responde buscando e analisando os dados de treinamento em uma "vizinhança local" de $x\_teste$.

Todos os algoritmos de aprendizagem baseada em memória envolvem dois ingredientes essenciais:

\begin{enumerate}

\item O critério utilizado para definir a vizinhança local do vetor de teste $x\_teste$;

\item a regra de aprendizagem aplicada aos exemplos de treinamento na vizinhança local de $x\_teste$.

\end{enumerate}

Em um tipo mais efetivo de aprendizagem baseada em memória, conhecido como a regra do vizinho mais próximo, a vizinhança local é definida como o exemplo de treinamento que se encontra na vizinhança imediata do vetor de teste $x\_teste$. Em particular, diz-se que o vetor

\begin{equation}

\label{eq:nearestneighborhood\_vector}

x'\_N \in \{x\_1, x\_2, ..., x\_N\},

\end{equation}

é o vizinho mais próximo de $x\_teste$ se

\begin{equation}

\label{eq:nearestneighborhood}

\min\_i d(x\_i, x\_teste) = d(x'\_N,x\_teste),

\end{equation}

onde $d(x\_i, x\_teste)$ é a distância euclidiana entre os vetores $x\_i$ e $x\_teste$. A classe associada com a distância mínima, ou seja, o vetor $x'\_N$ é apresentada como a classificação de $x\_teste$. Esta regra é independente da distribuição fundamental responsável pela geração dos exemplos de treinamento.

**A PARTIR DAQUI ESTÁ EM INGLÊS, VOU TRADUZIR AINDA. MAS BASICAMENTE, COPIEI O CONTEÚDO DE ALGUNS ARTIGOS.**

**Gradiente Conjugado Escalonado**

Conjugate gradient optimization trades off the simplicity of GD and the fast quadratic convergence of Newton's method. Several conjugate gradient learning algorithms have been developed, most of which are based on the assumption that the error function of all weights in the region of the solution can be accurately approximated by \cite{engelbrecht2007computational}

\begin{equation}

\label{eq:scg1}

\varepsilon\_T(D\_T,w) = \frac{1}{2}w^THw - \theta^Tw

\end{equation}

where $H$ is the Hessian matrix. Since the dimension of the Hessian matrix is the total number of weights in the network, the calculation of conjugate directions on the error surface becomes computationally infeasible. Computationally feasible conjugate gradient algorithms compute conjugate gradient directions without explicitly computing the Hessian matrix, and perform weight updates along these directions.

An important aspect in conjugate gradient methods is that of direction vectors, $\{p(0),p(1),...,p(t-1)\}$. These vectors are created to be conjugated with the weight vector, w. That is, $p^T(t\_1)wp(t\_2) = 0$ for $ t\_1 \neq t\_2$. A new conjugate direction vector is generated at each iteration by adding to the calculated current negative gradient vector of the error function a linear combination of the previous direction vectors. The standard conjugate gradient algorithm assumes a quadratic error function, in which case the algorithm converges in no more than $n\_w$ steps, where $n\_w$ is the total number of weights and biases. The Fletcher-Reeves conjugate gradient algorithm does not assume a quadratic error function. The algorithm restarts after $n\_w$ iterations if a solution has not yet been found \cite{engelbrecht2007computational}.

The scale factors can also be calculated through Polak-Ribiere and Hestenes-Stiefer methods. Moller \cite{moller1993scaled} proposed the scaled conjugate gradient (SCG) algorithm as a batch learning algorithm. Step sizes are automatically determined, and the algorithm is restarted after $n\_w$ iterations if a good solution was not found.

\subsubsection{Quasi-Newton Learning}

The derivatives in minimization algorithms are the gradient and the Hessian. The gradient must be known accurately as descent directions have to be calculated from them and approximations to gradient do not provide the required accuracy. On the other hand, the Hessian can be approximated by secant techniques. Since the Hessian is the Jacobian of the nonlinear system of equations $\nabla F(x) = 0$, it could be approximated. But, the Hessian has two important properties: it is always symmetric and often positive definite. The incorporation of these two properties into the secant approximation is an important aspect of the BFGS-BP and OSS-BP methods discussed subsequently. They are most often called positive definite secant updates\cite{saini2002artificial}.

\subsubsection{Broyden-Fletcher-Goldfarb-Shanno back-propagation method}

In Newton's method a quadratic approximation is used instead of a linear approximation of the function $F(x)$. The next approximate solution is obtained at a point that minimizes the quadratic function

\begin{equation}

\label{eq:bfgs1}

F(x\_{k+1}) = F(x\_k + \Delta x\_k) = F(x\_{k})+ g^T\_k\Delta x\_k + \frac{1}{2}\Delta x^T\_k A\_k \Delta x\_k

\end{equation}

Hence, the obtained sequence is

\begin{equation}

\label{eq:bfgs2}

x\_{k+1} = x\_k - A^{-1}\_k g\_k

\end{equation}

The main advantage of the Newton's method is that it has a quadratic convergence rate, while the steepest descent has a much slower, linear convergence rate. However, each step of Newton's method requires a large amount of computation. Assuming that the dimensionality of the problem is N, an O($N^3$) floating-point operation is needed to compute the search direction $d^k$. A method that uses an approximate Hessian matrix in computing the search direction is the quasi-Newton method.Let $H\_k$ be an $N \times N$ symmetric matrix that approximates the Hessian matrix $A\_k$; then the search direction for the quasi newton-Newton method is obtained by minimizing the quadratic function \cite{saini2002artificial}

\begin{equation}

\label{eq:bfgs3}

F(x\_{k+1}) = F(x\_{k})+ g^T\_k\Delta x\_k + \frac{1}{2}\Delta x^T\_k H\_k \Delta x\_k

\end{equation}

As the matrix $H\_k$ is to approximate the Hessian of the function $F(x)$ at $x = x\_k$, it needs to be updated from iteration to iteration by incorporating the most recent gradient information \cite{saini2002artificial}.

\subsubsection{One-step Secant back-propagation method}

One drawback of the BFGS update of Equation \ref{eq:bfgs3} is that it requires storage for a matrix of size $N \times N$ and calculations of order O($N^ 2$). Although the available storage is less of a problem now than it was a decade ago, the computational problem still exists for large $N$. It is possible to use a secant approximation with O($N$) computing. In this method, the new search direction is obtained from vectors computed

from gradients. If $g\_{k+1}$ is the current gradient, the new search direction, $p\_{k+1}$ is obtained as \cite{saini2002artificial}

\begin{equation}

\label{eq:oss1}

p\_{k+1} = - p\_{k+1} + B\_k y\_k +C\_k s\_k

\end{equation}

where $g\_{k+1} = \nabla F(x\_{k+1})$ and the two scalars $B\_k$ and $C\_k$ are the following combination of scalar products of the previously defined vectors $s\_k, g\_{k+1}$ and $y\_k$ (last step, current gradient and difference of gradients)

\begin{equation}

\label{eq:oss2}

B\_k = \frac{s^T\_k g\_{k+1}}{s^T\_k y\_k}

\end{equation}

and

\begin{equation}

\label{eq:oss2}

C\_k = 1 \left(1 + \frac{y^T\_k y\_k}{s^T\_k y\_k} \right) \frac{s^T\_k g\_{k+1}}{s^T\_k y\_k} + \frac{y^T\_k g\_{k+1}}{s^T\_k y\_k}

\end{equation}

The backtracking line search is used with the quasi-Newton's OSS optimization algorithm. At the beginning of learning, the search direction is $-g\_0$ and it is restarted to $-g\_k+1$ every $N$ steps. Multiplier 1.1 increases the last successful learning rate and first tentative step is executed. If the energy of the network is higher than the upper limiting value, then a new tentative is tried by using successive quadratic interpolation until the requirement is met. The learning rate is decreased to half after each unsuccessful trial \cite{saini2002artificial}.

\subsubsection{Levenberg-Marquardt Algorithm}

The Levenberg-Marquardt algorithm adaptively varies the parameter updates between the gradient descent update and the Gauss-Newton update \cite{marquardt1963algorithm},

\begin{equation}

\label{eq:scg}

\varepsilon\_T(D\_T,w) = \frac{1}{2}w^THw - \theta^Tw

\end{equation}

where small values of the algorithmic parameter $\lambda$ result in a Gauss-Newton update and large values of $\lambda$ result in a gradient descent update. The parameter $\lambda$ is initialized to be large. If an iteration happens to result in a worse approximation, $\lambda$ is increased. As the solution approaches the minimum, $\lambda$ is decreased, the Levenberg-Marquardt method approaches the Gauss-Newton method, and the solution typically converges rapidly to the local minimum \cite{marquardt1963algorithm}.

Marquardt’s suggested update relationship in Equation \ref{eq:LMupdate}.

\begin{equation}

\label{eq:LMupdate}

\varepsilon\_T(D\_T,w) = \frac{1}{2}w^THw - \theta^Tw

\end{equation}

\pagebreak