Universidade Federal de Santa Catarina
Departamento de Automação e Sistemas

Introdução à Inteligência Computacional

Eduardo Camponogara

Março de 2006

Sumário

Capítulo 1

Inteligência Computacional

"Computational intelligence is the study of the design of intelligent agents." [David Poole, Alan Mackworth, and Randy Goebel, 1998]

1.1 O que é inteligência computacional?

Inteligência computacional é o estudo e projeto de agentes inteligentes. Um *agente* é alguma coisa que atua no ambiente — ele faz alguma coisa. Agentes incluem vermes, cachorros, termostatos, aviões, homens, organizações e a sociedade. Um *agente inteligente* é um sistema que age de forma inteligente: o que ele faz é apropriado para as circunstâncias e objetivos, é flexível com respeito a ambientes e objetivos variantes no tempo, aprende a partir da experiência, e toma as ações apropriadas dados a limitação sensorial e a capacidade computacional.

O objetivo científico da inteligência computacional é o entendimento dos princípios que induzem comportamento inteligente, em sistemas naturais e artificiais. A hipótese principal é que raciocínio equivale à computação. O objetivo mais prático é a especificação de métodos para projeto de artefatos inteligentes e úteis.

1.2 Inteligência artificial ou computacional?

Inteligência artificial (IA) é o nome da disciplina estabelecida para inteligência computacional (IC), mas o termo *"inteligência artificial"* é fonte de discordância e confusão. Podemos dizer que inteligência artificial é inteligência verdadeira? Talvez não, da mesma forma que uma pérola de imitação não é uma pérola verdadeira. "*Inteligência sintética*" poderia ser uma alternativa mais razoável, como uma pérola sintética não é natural mas mesmo assim é uma pérola verdadeira. Como o objetivo central está no entendimento de sistemas naturais e artificiais (sintéticos), preferimos o termo *"inteligência computacional"*. Este termo ainda tem a vantagem de fazer a hipótese computacional explícita.

A confusão sobre o nome da disciplina pode, em parte, ser atribuída a uma fusão do propósito da disciplina com sua metodologia. O propósito é entender como comportamento inteligente é possível. A metodologia está no projeto, construção e experimentação de sistemas computacionais capazes de executar tarefas tipicamente

vistas como inteligentes. A construção destes artefatos é uma atividade essencial já que inteligência computacional é, acima de tudo, uma ciência empírica; contudo não pode ser confundida com seu propósito científico.

Outra razão para remover o adjetivo *"inteligente"* advém da conotação de inteligência simulada. Contrário a uma má interpretação, o objetivo não é simular comportamento inteligente. O objetivo é entender sistemas inteligentes (naturais ou sintéticos) por meio da sintetização destes sistemas. A simulação de um terremoto não é um terremoto; no entanto, desejamos criar entidades inteligentes, da mesma forma que você poderia imaginar a criação de um terremoto. A confusão tem como raiz a realização de simulações em computadores. Entretanto, devemos ter computadores digitais como sistemas de interpretação automática, formal e de manipulação simbólica que são ferramentas especiais: eles podem produzir coisas reais.

O agente inteligente óbvio é o ser humano. Muitos de nós acreditam que cachorros são inteligentes, mas não diríamos que vermes, bactérias ou insetos são inteligentes. Existe uma classe de agentes que são mais inteligentes que o ser humano, a classe das *organizações*. Colônias de formigas são exemplos típicos de organizações. Cada indivíduo tem inteligência limitada, mas uma colônia de formigas pode atuar de forma mais inteligente que qualquer formiga. A colônia pode descobrir comida e explorar a reserva de forma mais eficaz bem como se adaptar a situações variantes. De maneira similar, empresas podem desenvolver, manufaturar e distribuir produtos de maneira que a soma das habilidades necessárias é muito maior do que a habilidade de qualquer indivíduo. Computadores modernos, do nível mais baixo do hardware ao nível mais alto de software, são mais complicados do que pode ser entendido por qualquer ser humano, apesar disto eles são fabricados diariamente por organizações de seres humanos. A sociedade humana, vista como um agente, é possivelmente o agente mais inteligente conhecido.

1.3 Máquinas voadoras e máquinas pensantes

É instrutivo considerar a analogia entre o desenvolvimento de máquinas voadoras durante os últimos séculos e o desenvolvimento de máquinas *pensantes* nas últimas décadas.

Primeiro, note que existem diversas maneiras de se entender a noção de voar. Uma maneira consiste em dissecar um pássaro e fazer hipóteses sobre as características estruturais necessárias para um agente voador. Com este método de investigação de pássaros, morcegos e insetos poderíamos sugerir que a habilidade de voar envolve o bater de asas por uma estrutura coberta de penas ou uma membrana. Além disso, a hipótese poderia ser verificada envolvendo penas nos braços, batendo os braços e saltando. Poderíamos ainda imaginar um pesquisador afirmar que uma estrutura com camadas de penas é suficiente para desenvolver a habilidade de voar.

Uma metodologia alternativa está no entendimento do vôo sem nos restringirmos às ocorrências naturais. Isto envolve a construção de artefatos em formas diferentes das de um pássaro. Este método forneceu uma ferramenta útil para o entendimento dos princípios que regem a habilidade de voar, conhecidos como *aerodinâmica*, e que levaram à construção de aviões.

Esta diferença distingue inteligência computacional de outras disciplinas científicas da área cognitiva. Os cientistas do domínio de IC estão interessados em testar hipóteses gerais sobre a natureza da inteligência por meio da construção de máquinas que são inteligentes e que não imitam simplesmente o comportamento de homens e organizações. Isto nos permite uma outra maneira de tratar a questão *"Computadores podem realmente pensar?"* ao consideramos a questão análoga *"Aviões podem realmente voar?"*.

1.4 Modelos da mente

Através dos tempos seres humanos fizeram uso de tecnologia para modelar a si próprios. A parábola Taoísta do livro de *Lie Tzu* descreve um robô que se comportava como um ser humano, demonstrando habilidades sensoriais e interação que não eram diferentes das de um homem. No Egito e Grécia Antiga também há exemplos deste tipo de atividade. Cada nova tecnologia foi utilizada para construir agentes inteligentes ou modelar a mente. Relógios, hidráulica, sistemas de telefonia, hologramas e computadores analógicos foram propostos como metáforas tecnológicas de inteligência e como mecanismos para modelar a mente.

A partir destas ocorrências podemos especular que uma equivalência funcional não implica em uma equivalência estrutural. Para produzir comportamento inteligente não é necessário reproduzir as conexões estruturais do corpo humano.

Isto traz à tona a indagação se computadores digitais são apenas mais uma metáfora tecnológica, talvez a ser superada por outros mecanismos. Em parte, uma resposta deve ser empírica. Teremos de aguardar resultados substanciais para se chegar a uma resposta. Contudo, há razões para se acreditar que a resposta é *"não"*. Algumas razões são empíricas: os resultado obtidos até os dias atuais são significativos, mas não são conclusivos. Há outras razões. Considere as duas hipóteses abaixo. A primeira é conhecida por *hipótese símbolo-sistema*:

> *Raciocínio é manipulação de símbolos.*

A segunda hipótese é chamada *hipótese de Church-Turing* [1]:

> *Qualquer manipulação simbólica pode ser realizada em uma máquina de Turing.*

A máquina de Turing é um modelo abstrato do computador digital com memória ilimitada. Esta hipótese implica que qualquer manipulação simbólica, e portanto raciocínio, pode ser realizada em computadores digitais.

Não há como provar estas hipóteses matematicamente. Tudo que pode ser feito é empiricamente testá-las através da construção de sistemas de raciocínio. Por quê deveríamos acreditar que as hipóteses são razoáveis e até mesmo verdadeiras? A razão é que a comunicação verbal (linguagem), uma das poucas formas de comunicação com a mente, consiste da transmissão de símbolos. Raciocínio tem símbolos como entradas e saídas em termos de linguagem e, portanto, a função entre entrada e saída também pode ser descrita simbolicamente e pode ser implementada por meio da manipulação simbólica. Além disso, a inteligência encontrada na sociedade e nas

organizações é transmitida por meio de linguagens e símbolos. Tão logo algo seja expresso na linguagem, raciocínio sobre o que foi expresso consiste de manipulação simbólica. Estas hipóteses não nos dizem como implementar raciocínio em um certo computador — esta é a tarefa da inteligência computacional. Elas nos dizem que a computação é uma metáfora adequada para raciocínio.

Estaa hipótesea não afirmam que cada detalhe de computação deve ser interpretado simbolicamente. Elas também não dizem que cada instrução de máquina em um computador, ou a função de cada neurônio do cérebro, pode ser interpretado simbolicamente. Elas nos dizem que existe um nível de abstração no qual raciocínio pode ser interpretado com manipulação simbólica, e que este nível pode explicar as ações de um agente com base nas entradas.

Antes de aceitar estas hipóteses, podemos ainda considerar como elaa podem estar erradas. Uma alternativa é assumir que as ações são provenientes de uma função contínua das entradas para um agente tal que os valores intermediários não necessariamente correspondem a algo com significado. É até mesmo possível que a funcionalidade não seja interpretada simbolicamente, sem a necessidade de recorrer ao uso de números sem significado. Alternativas estão sendo perseguidas em redes neurais e robôs reativos inspirados em insetos artificiais.

1.5 Ciência e engenharia

Como sugerido pela analogia com máquinas voadoras, existe uma "*tensão*" entre a ciência da inteligência computacional (que tenta entender os princípios por trás do raciocínio) e a engenharia da inteligência computacional (que constrói programas para resolver problemas particulares). Esta tensão é uma parte essencial desta disciplina.

Como IC é uma ciência, a literatura neste domínio deve ser manifestada por meio de métodos científicos, especialmente a criação e teste de teorias refutáveis. Questões óbvias são *"Do que tratam as teorias de IC?"* e "*Como se testaria uma teoria caso se conhecesse alguma?"*. Teorias de IC têm a ver com problemas interessantes que podem ser representados e resolvidos por computador. Teorias são reforçadas empiricamente através da construção de implementações, que são avaliadas por princípios tradicionais de ciência da computação. Não é possível realizar inteligência computacional sem especificar teorias e construir protótipos; eles são inexoravelmente conexos. É claro que um cientista não precisa realizar ambos, mas ambos precisam ser realizados. Um experimento não tem significado algum sem uma teoria a ser avaliada, e uma teoria sem evidência no sentido de suportá-la ou refutá-la é de pouco valor. O princípio de *"Ockham's Razor"* é o guia mestre: sempre prefira teorias simples e implementações menos complexas.

Com este pano de fundo, podemos considerar uma das questões mais recorrentes no campo da inteligência computacional: *"o comportamento humano é algorítmico?"*. Não precisamos tentar responder esta questão ao aceitarmos que a resposta não é conhecida, e que uma resposta é o objetivo das ciências cognitivas e da inteligência computacional.

1.6 Relações com outras disciplinas

A inteligência computacional é uma disciplina jovem. Outras disciplinas diversas, incluindo filosofia, biologia evolucionária, ciência política, engenharia de controle e outras, também vêm estudando inteligência há mais tempo. Inicialmente, podemos discorrer sobre a relação da filosofia e psicologia com o estudo da inteligência, para então depois discutir a relação com a ciência da computação.

A inteligência computacional poderia ser descrita como *"psicologia sintética"*, *"filosofia experimental"* ou *"epistemologia computacional"* — *epistemologia* é o estudo do conhecimento. A IC pode ser vista como uma forma de estudar o velho problema da natureza do conhecimento e da inteligência, mas com um método experimental muito mais poderoso do que o disponível no passado. Em vez de se observar apenas o comportamento externo de sistemas inteligentes, como a filosofia, psicologia, economia e sociologia tradicionalmente fizeram, somos agora capazes de realizar experimentos com modelos de comportamento inteligente. Tais modelos estão abertos à inspeção, podem ser reprojetados e submetidos à experimentação de forma completa e rigorosa. Em palavras, agora podemos construir modelos que os filósofos só podiam discutir no nível teórico. Agora podemos experimentar como estes modelos e não apenas discutir propriedades abstratas. As teorias podem ser fundamentadas empiricamente através de experimentação.

Da mesma forma que o objetivo da aerodinâmica não é sintetizar pássaros, mas de entender os fenômenos relacionado à habilidade de voar através da construção de máquinas voadoras, o objeto último da inteligência computacional não é necessariamente a simulação em larga-escala da inteligência humana. A noção de validade psicológica separa o trabalho em IC em duas categorias: um que se preocupa em imitar a inteligência, freqüentemente chamado de *"modelagem cognitiva"*; e outro que não se preocupar em imitá-lo.

Para enfatizar o desenvolvimento da inteligência computacional como ciência, não estamos preocupados com a validade psicológica, mas como o desejo prático de criar programas que resolvem problemas reais. Algumas vezes será importante que o computador *"raciocine"* sobre um problema de forma assemelhada ao raciocínio humano. Isto é essencialmente importante quando desejamos uma explicação de como se chegou à resposta. Alguns aspectos da cognição humana talvez não devam ser replicados como, por exemplo, a tendência de cometer erros de aritmética.

Inteligência computacional está relacionada com a disciplina de ciência da computação. Enquanto há vários cientistas fora da ciência da computação que conduzem pesquisas em IC, a maior parte dos pesquisadores em IC (ou IA) é membro de departamentos de ciência da computação. Acreditamos que isto seja apropriado visto que o estudo da computação é central para a inteligência computacional. O entendimento de algoritmos, estruturas de dados e complexidade combinatória são essenciais para a construção de máquinas inteligentes. É também surpreendente o quanto a ciência da computação se beneficiou de trabalhos oriundos da inteligência artificial, desde as idéias de tempo compartilhado até sistemas de álgebra computacional.

Há outros campos cujo objetivo é a construção de máquinas que agem de forma inteligente. Dois destes campos são a engenharia de controle e a pesquisa operacional. Estes campos tem raízes em pontos diferentes da IC, especificamente o uso

de matemática contínua. Visto que a construção de agentes inteligentes envolve o controle contínuo e o raciocínio do tipo IC, estas disciplinas devem ser vistas com simbióticas com a IC. Um estudante de qualquer destas disciplinas deve ser capaz de entender a outra. Além disso, a distinção entre elas está se tornam menos clara com o desenvolvimento de teorias que combinam diferentes áreas. Infelizmente existe pouco material sobre engenharia de controle e pesquisa operacional, apesar de que muitos resultados foram desenvolvidos em pesquisa operacional e IC.

Por fim, a inteligência computacional pode ser vista como um *"chapéu"* para as ciências cognitivas. Ciências cognitivas ligam várias disciplinas do estudo da cognição e do raciocínio, incluindo a filosofia, lingüística, antropologia e neurociência. A inteligência computacional se distingue dentro das ciências cognitivas porque esta provê ferramentas para construção de entidades inteligentes, não apenas se preocupando com o estudo do comportamento externo de agentes inteligentes ou com a dissecação de sistemas inteligentes para entender o funcionamento destes.

1.7 Agentes

Há várias questões filosóficas de interesse sobre a natureza da inteligência computacional, mas para entendermos como o comportamento inteligente pode ser algorítmico devemos tentar programar o computador para resolver problemas concretos. Não é suficiente especular que um comportamento interessante e particular é algorítmico. Há que se desenvolver uma teoria que explica como este comportamento se manifesta nas máquinas e, então, mostrar como tal teoria pode ser realizável por meio da construção de máquinas que implementam tal teoria.

Estamos interessados no raciocínio prático: raciocínio que faça algo de interesse, útil. A integração de habilidades de percepção, raciocínio e atuação constitui um *agente*. Um agente poderia ser, por exemplo, a agregação de uma máquina computacional com sensores e atuadores físicos — um *robô*. Poderia originar-se da agregação de um computador que provê recomendações — um *sistema especialista* — com um especialista humano que alimenta o computador com informações sensoriais e conduz o diálogo. Um agente poderia ainda ser um programa que age em um ambiente puramente computacional.

A Fig. 1.1 mostra as entradas e saídas de um agente. A qualquer instante o agente tem:

- conhecimento anterior sobre o mundo (ambiente);
- experiência passada que pode servir de base para aprendizagem;
- objetivos que tentar alcançar, por meio da maximização de valores associados com o que é importante; e
- informações sobre o estado corrente do ambiente e si próprio.

Para cada agente procuramos definir a forma das entradas e das ações. O objetivo do estudo nesta disciplina está na implementação da *caixa-preta* tal que ações satisfatórias são executadas dadas as entradas.

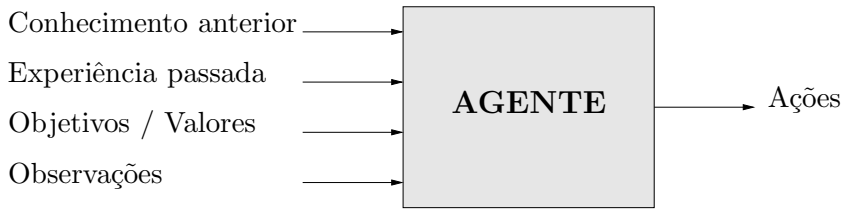

Figura 1.1: Agente como caixa-preta.

Para o propósito do estudo daqui a diante, o mundo consiste de um agente inserido em um ambiente. O ambiente pode até mesmo incluir outros agentes. Cada agente pode ter seus próprios objetivos (a serem perseguidos), maneiras de influenciar o ambiente na busca de seus objetivos, mecanismos para modificar suas crenças através de raciocínio, habilidade de percepção e aprendizagem. Esta é uma visão completa de sistemas inteligentes que variam desde termostatos até um time de robôs móveis. O sucesso na construção de agentes inteligentes depende, é claro, do problema que foi selecionado para investigação. Alguns problemas são bem-adequados para uso de computadores, tal como a ordenação de números. Outros problemas podem não ser, tal como trocar as fraldas de uma criança ou propor uma boa estratégia política.

É importante avaliar a literatura contemporânea de inteligência computacional, antecipar o futuro e desenvolver conceitos e ferramentas que permitam a construção, teste e modificação de agentes. Na continuidade deste texto, estudaremos formalismos que podem ser empregados, com graus de eficiência variados para problemas diferentes, na construção de agentes inteligentes. Estes formalismos incluem as redes neurais artificiais, os algoritmos genéticos e a lógica fuzzy (ou nebulosa), que serão objeto de estudo neste texto.

1.8 Referências

O texto apresentado acima é uma compilação traduzida extraída do livro de Poole *et al.* [13]. Outro material de referência é o livro de Russel e Norvig [14].

Capítulo 2

Redes Neurais

"A neural network is an interconnected assembly of simple processing elements, units or nodes, whose functionality is loosely based on the animal neuron." [Kevin Gurney, 1999]

2.1 Introdução

Redes neurais artificiais são modelos extremamente simplificados das redes neurais biológicas, construídas a partir da interconexão de unidades de processamento (neurônios, elementos de processamento, ou nós) que manipulam sinais de entrada e produzem sinais de saída para outras unidades. A partir de uma base de pares de entrada e saída de uma função desconhecida, $\{(x^p, y^p)\}$, podemos treinar a rede neural a reproduzir a relação $y = f(x)$ com $f_w(x) \approx f(x)$, onde $f_w(x)$ é a saída da rede neural para a entrada x e w é o vetor dos parâmetros da rede, minimizando o erro $y^p - f_w(x^p)$ entre a saída desejada e a produzida pela rede. Redes neurais (artificiais) constituem métodos *"robustos"* de aproximação de funções de valores vetoriais, reais ou discretos. Em outras palavras, as redes neurais toleram erros nos pares de treinamento. Uma rede neural pode ser vista como um aproximador universal ou, mais precisamente, como um interpolador universal uma vez que elas não são capazes de inferir informações que estão fora do conjunto de treinamento. Dado um conjunto qualquer de treinamento podemos, em princípio, construir uma rede neural que produz os mesmos valores da função para os pares entrada-saída de treinamento. O algoritmo de treinamento de propagação reversa (*back-propagation*) permite realizar o treinamento de redes neurais que geram funções não-lineares, sendo amplamente utilizado.

Neste capítulo faremos um breve estudo dos princípios biológicos por trás das redes neurais, dos modelos de redes neurais, algoritmos de treinamento e aplicações.

2.2 Cérebro: um sistema de processamento de informações

O cérebro humano contém cerca de 10 bilhões de células neurais ou neurônios. Em média, cada neurônio está conectado a outros neurônios por meio de aproxima-

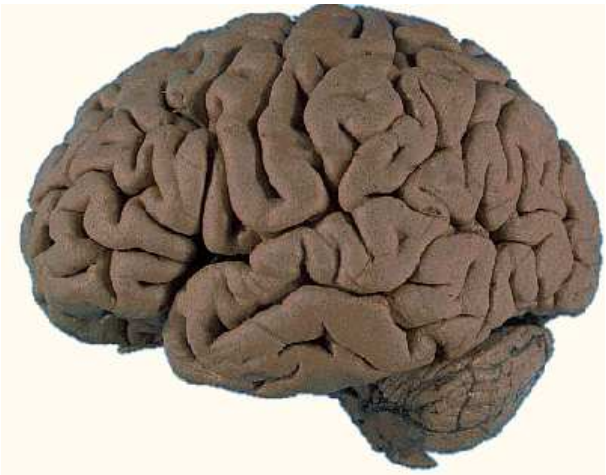

Figura 2.1: Cérebro humano.

damente 10.000 sinapses. A rede de neurônios forma um sistema de processamento de informações massivamente paralelo, contrastando com computadores convencionais que executam uma instrução a cada ciclo. Por outro lado, considere o tempo de execução de uma operação elementar: neurônios operam a uma freqüência típica de 100 Hz, enquanto CPUs realizam centenas de milhões de operações por segundo. Apesar de ser construído a partir de "*hardware*" lento, o cérebro possui capacidades surpreendentes:

- o desempenho decai gradativamente com danos parciais; por outro lado, muitos programas e sistemas de engenharia são frágeis: se removemos um componente arbitrário, é bem possível que o sistema como um todo falhe.

- ele pode aprender e se organizar a partir da experiência adquirida; isto significa que recuperação parcial é possível após danos se as unidades saudáveis podem aprender a assumir as funções previamente sob responsabilidade das unidades danificadas.

- ele realiza computações massivamente paralelas de forma muito eficiente; por exemplo, o processo complexo de percepção visual ocorre em menos de 100 ms, o que corresponde a 10 passos de processamento.

Como parte da inteligência computacional, redes neurais constituem um formalismo que busca dotar computadores de capacidades do cérebro humano por meio da imitação de certos aspectos do processamento neural de informações, de maneira extremamente simplificada.

Redes neurais

O cérebro não é homogêneo (Fig. 2.1). Na escala anatômica (maior escala), podemos distinguir o CORTEX, CEREBÊLO, MIDBRAIN e BRAINSTEM. Cada um desses elementos anatômicos pode ser subdividido em áreas dentro de cada região, de acordo com a estrutura anatômica da rede neural ou conforme a função executada

Tabela 2.1: Comparação cérebro e computador

	cérebro	computador convencional
elementos de processamento	10^{14} sinapses	10^{8} transistores
tamanho do elemento	10^{-6} m	10^{-6} m
uso de energia	30 W	30 W (CPU)
velocidade de processamento	100 Hz	10^{9} Hz
estilo de processamento	paralelo, distribuído	serial, centralizado
tolerância a faltas	sim	não
aprendizado	sim	pouco
consciência	usualmente	ainda não

por elas. O padrão das conexões (padrão de projeção) entre as áreas é extremamente complexo e conhecido apenas parcialmente. A Fig. 2.2 ilustra processos cognitivos associados a diferentes regiões do cérebro. O maior e mais bem conhecido sistema do cérebro humano é o sistema visual, que se conhece 10 dos 11 estágios de processamento. Podemos discernir entre projeções FEEDFORWARD que partem dos estágios iniciais de processamento anterior (próximos dos sensores de entrada) para os estágios terminais (próximo das conexões de saída), e projeções FEEDBACK que vão na direção oposta.

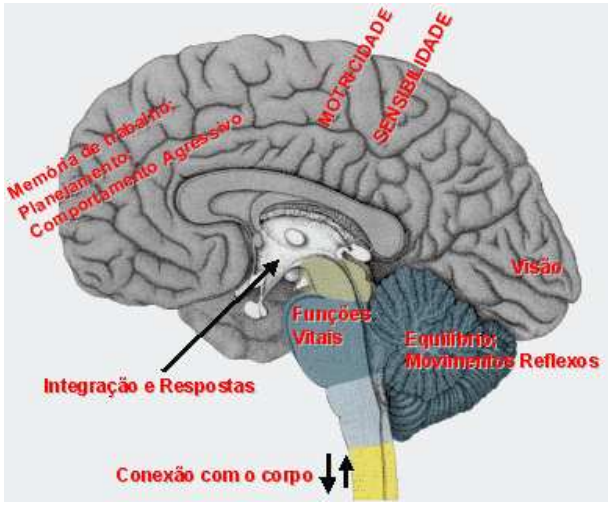

Figura 2.2: Regiões do cérebro.

Além destas conexões de longo alcance, neurônios estão conectados com milhares de outros neurônios na sua vizinhança, formando redes neurais locais complexas e densas, conforme ilustra a Fig. 2.3.

Neurônios e sinapses

A unidade computacional básica do sistema neural é a célula nervosa ou neurônio (Fig. 2.4). O neurônio é formado por:

- dendritos (entradas)

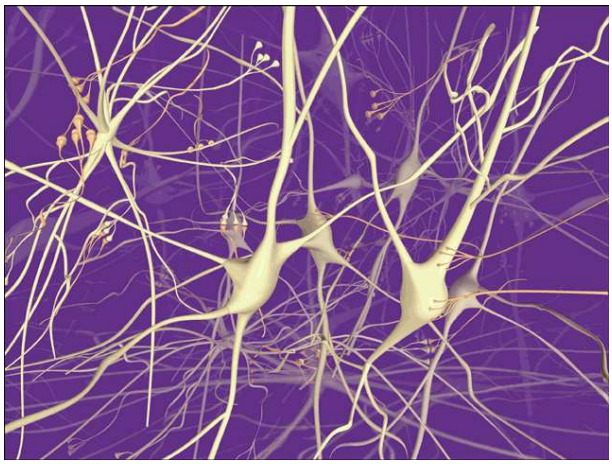

Figura 2.3: Rede neural biológica.

- corpo da célula
- axônio (saída)

Um neurônio recebe entrada de outros neurônios, tipicamente vários milhares de neurônios. As entradas são APROXIMADAMENTE somadas e quando esta soma excede um certo nível crítico, o neurônio libera um impulso — um sinal elétrico que viaja através do corpo, depois através do axônio e até os próximos neurônios. O fenômeno de disparo de impulso também é conhecido por DESPOLARIZAÇÃO, sendo seguido por um período durante o qual o neurônio não é capaz de disparar. As terminações do axônio (zona de saída) quase tocam os dendritos ou corpo da célula do próximo neurônio. A transmissão do sinal elétrico de um neurônio para o próximo é afetada por neuro-transmissores, substâncias químicas liberadas pelo primeiro neurônio que *"casam"* com os receptores do segundo. Este comportamento é conhecido como sinapse. O grau no qual o sinal enviado por um neurônio é passado à frente depende de vários fatores, dentre eles podemos destacar a quantidade de neuro-transmissores disponíveis, o número e organização de receptores, e a quantidade de neuro-transmissores reabsorvidos.

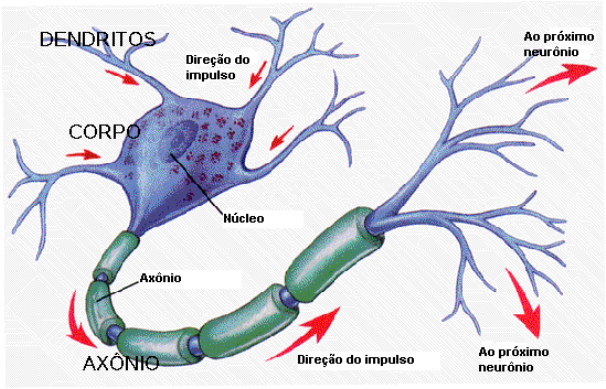

Figura 2.4: Estrutura de um neurônio.

Aprendizado sináptico

O cérebro é capaz de aprender. Baseado no conhecimento contemporâneo da estrutura neuronal, uma forma do cérebro aprender é por meio da alteração, adição e remoção de conexões sinápticas entre neurônios. O cérebro aprende ON-LINE com base na experiência acumulada e tipicamente sem o benefício de um instrutor. A eficácia de uma conexão sináptica pode mudar com a experiência, caracterizando memória e aprendizado através da potenciação de longo-prazo (PLP). Uma forma se dá através da liberação de neuro-transmissores, mas muitas outras formas existem. A potenciação de longo-prazo é definida como o aumento persistente de mais de uma hora na eficácia de uma conexão sináptica que resulta da estimulação de alta freqüência de um caminho de entrada.

Hebbs postulou:

> *"Quando o axônio de uma célula A ... excita uma célula B e, repetidamente e persistentemente, influencia o disparo desta, um processo de crescimento e modificação metabológica ocorre em ambas as células de forma que a influência de A como uma das células que causa o disparo de B aumenta."*

Bliss e Lomo descobriram PLP no Hipocampo em 1973.

2.3 Modelo do neurônio artificial

Neuro-cientistas elaboraram um modelo computacional do neurônio que permite realizar simulação detalhada de um circuito específico do cérebro. Do lado da ciência da computação, estamos mais interessados nas propriedades gerais de uma rede neural, independentemente de como elas são implementadas no cérebro. Isto nos permite utilizar modelos mais simples, *"neurônios abstratos"*, que podem capturar a essência da computação neural mesmo quando muitos detalhes sobre o neurônio biológico são ignorados. Modelos de neurônios foram implementados em hardware e até mesmo em circuitos VLSI (*very large scale integration*). Uma vez que computadores executam operações rapidamente, podemos simular o funcionamento de redes muito grandes montadas a partir de modelos simples. Isto nos permite realizar experimentações em software, sem a necessidade de hardware específico.

Um neurônio artificial simples

O modelo computacional básico (modelo de neurônio) é também chamado de unidade, como ilustra a Fig. 2.5. Ela recebe entradas de outras unidades ou de uma fonte externa. Cada entrada tem associado um peso w que pode ser modificado para modelar o aprendizado sináptico. A unidade i computa uma função f_i da soma ponderada das entradas:

$$y_i = f_i(\sum_{j=1}^{n} w_{ij} y_j)$$

A saída y_i da unidade i pode servir de entrada para outras unidades. A soma ponderada $\sum_{j=1}^{n} w_{ij}y_j$ é chamada the *"entrada líquida"* para a unidade i, rotineiramente denotada por net_i. Observe que w_{ij} se refere ao peso da influência que a unidade j exerce sobre i. A função f_i é a *"função de ativação"*. No caso mais simples, f_i é a função identidade fazendo com que a saída seja simplesmente a entrada líquida. Tal unidade é conhecida como *"unidade linear"*.

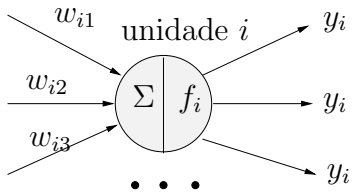

Figura 2.5: Estrutura de um neurônio.

2.4 Regressão linear

Considere o conjunto de pontos apresentados na Fig. 2.6. Cada ponto da figura dá informação sobre o peso (eixo x, em libras) e o consumo de combustível (eixo y, em galões por milha) para um certo veículo dentre 74 veículos. O peso e o consumo estão relacionados: quanto mais pesado o veículo, maior o consumo de combustível. Agora suponha que seja dado o peso do veículo de número 75, para o qual devemos predizer/estimar o consumo de combustível tomando como base os dados fornecidos. Tais questões podem ser respondidas por meio de um modelo — uma descrição matemática aproximada — dos dados. O modelo mais simples que ainda seja útil é da forma:

$$y = w_1 x + w_0 \tag{2.1}$$

O modelo acima é linear: no plano $x - y$, a equação (2.1) é uma linha com inclinação w_1 que intercepta o eixo y na altura w_0, como mostra a Fig. 2.7. Para esta reta $w_0 = 42$ e $w_1 = -0,0051923$. Quais valores para w_0 e w_1 devem ser utilizados? Qualquer linha traçada através dos pontos amostrais pode ser usada como um preditor, mas certas linhas incorrerão erros menores que outras. A linha dada na Fig. 2.7 não é um bom modelo: para muitos carros, a predição de consumo será exagerada para um certo peso.

Função erro

Para quantificar um "*bom preditor*" podemos definir uma função erro E (ou função objetivo) sobre os parâmetros do modelo. Uma forma popular da função erro é a soma dos erros quadráticos:

$$E = \frac{1}{2}\sum_i (\text{dado}_i - \text{predição}_i)^2 \tag{2.2}$$

$$= \frac{1}{2}\sum_i (t_i - y_i)^2 \tag{2.3}$$

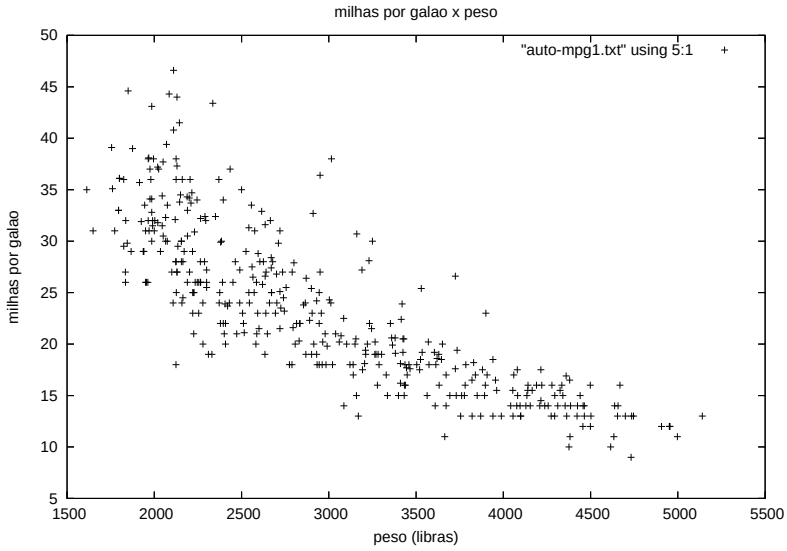

Figura 2.6: Milhas por galão como uma função do peso do carro.

Esta função é a soma sobre todos os pontos i dos dados amostrais da diferença quadrada entre o valor desejado t_i (consumo de combustível) e a predição y_i, calculada a partir da entrada x_i (o peso do veículo i) conforme a equação (2.1). No caso de um modelo linear, a soma dos erros quadráticos é uma função quadrática dos parâmetros do modelo. A Fig. 2.8 mostra a superfície da função erro, $E(w_0, w_1)$, para uma faixa de valores de w_0 e w_1. A Fig. 2.9 mostra a mesma função na forma de curvas de nível.

Minimizando o erro de predição

A função erro E dá uma medida do erro de predição para uma escolha específica de parâmetros. Podemos colocar o objetivo de encontrar o melhor modelo (linear) como o problema de encontrar os parâmetros que minimizam E. No caso de modelos lineares, a regressão linear nos dá uma forma direta de computar os parâmetros ótimos. Por outro lado, a solução analítica obtida não pode ser generalizada para modelos não-lineares (que serão objeto de estudo mais à frente). Apesar de não sermos capazes de calcular de forma explícita os parâmetros ótimos de modelos não-lineares, podemos aplicar métodos iterativos tais como o método de descenso [11]. Este método funciona da seguinte forma:

1. escolha valores iniciais para os modelos, possivelmente de forma aleatória.
2. calcule o gradiente g da função erro com respeito aos parâmetros modelo.
3. modifique os parâmetros através de um deslocamento pequeno na direção com maior taxa de decrescimento da função erro, isto é, na direção $-g$.
4. repita os passos 2 e 3 até que o gradiente chegue próximo de zero.

De que forma o método se comporta? O gradiente de E é a direção na qual a função erro tem a maior inclinação com respeito aos valores correntes w dos

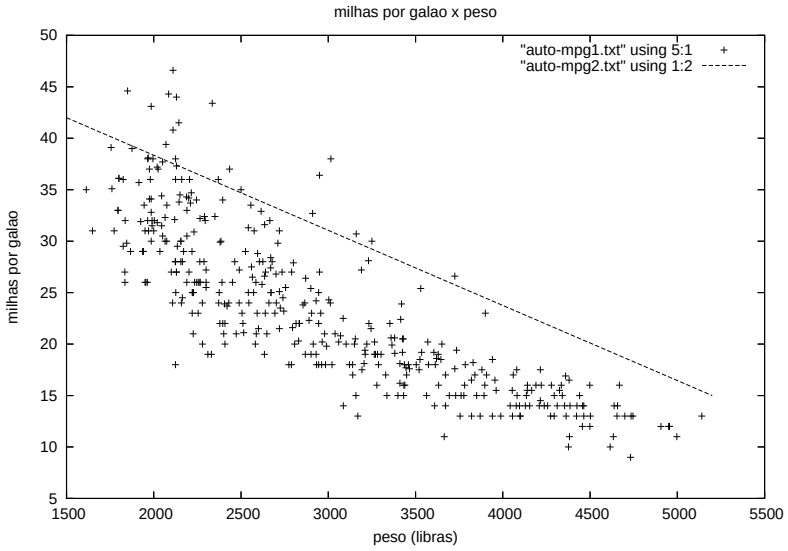

Figura 2.7: Aproximação linear da função milhas por galão.

parâmetros. Objetivando decrescer E, tomamos um passo pequeno na direção oposta ao gradiente, $-g$ (Fig. 2.10). A repetição destes passos move E "*ladeira abaixo*" até que se atinja um mínimo, onde $g = 0$, e não mais podemos reduzir o valor de E (Fig. 2.11). A Fig. 2.12 mostra a melhor reta para os dados fornecidos encontrados com este procedimento.

O modelo é uma rede neural

O modelo linear da equação (2.1) pode ser implementado como uma rede neural simples, conforme ilustra a Fig. 2.13. Esta rede consiste de um valor constante (*"bias"*), uma unidade de entrada e uma unidade linear de saída. A unidade de entrada faz o papel da entrada x (o peso do carro) que alimenta a rede, enquanto a unidade de *"bias"* tem como saída o valor 1. A unidade de saída computa a soma:

$$y_2 = y_1 w_{21} + 1.0 w_{20} \tag{2.4}$$

É imediata a verificação da equivalência entre a equação (2.1) com w_{21} que implementa a inclinação da linha e w_{20} que nos dá o ponto onde a reta intercepta o eixo y.

2.5 Redes neurais lineares

Regressão múltipla

O exemplo dos carros mostrou como podemos descobrir uma função linear para predição em uma variável (consumo de combustível) a partir de outra variável (peso). Suponha que agora são dados variáveis adicionais que podem ser úteis como preditores. O modelo neural simples (Fig. 2.13) pode ser estendido pela adição de outras unidades de entrada como indica a Fig. 2.14.

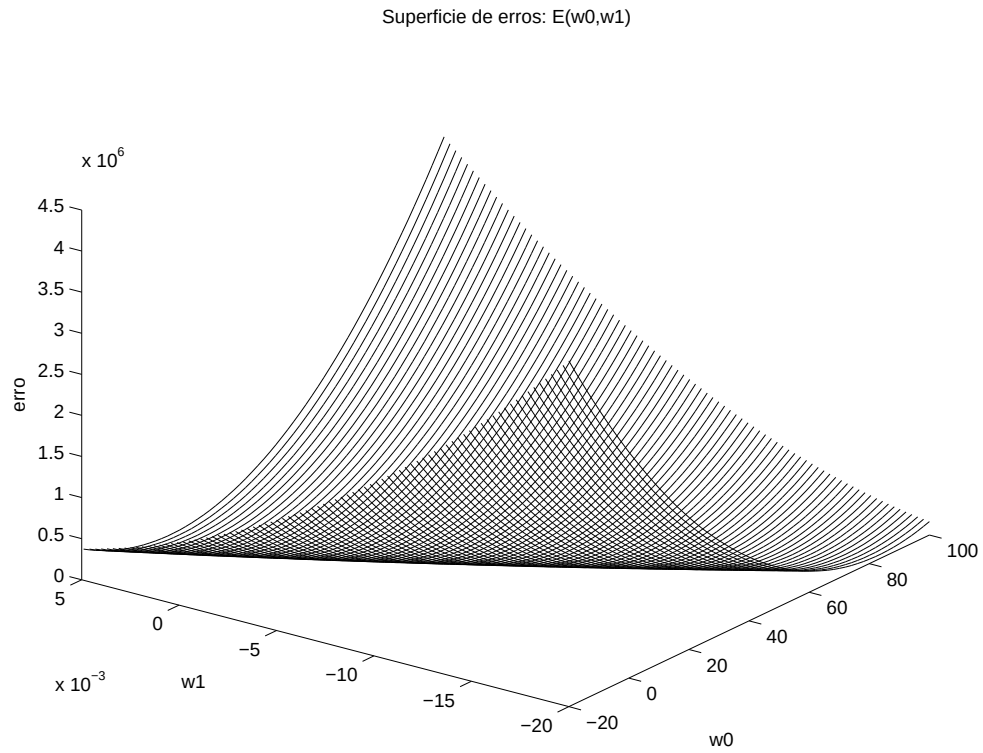

Figura 2.8: Superfície do erro $E(w_0, w_1)$.

Além disso, podemos estar interessados na predição de mais de uma variável dos dados fornecidos. Isso pode ser facilmente acomodado adicionando unidades de saída adicionais. Veja ilustração dada pela Fig. 2.15. A função de erro para uma rede com múltiplas saídas é obtida simplesmente adicionando o erro para cada unidade de saída. A rede passa a ter uma estrutura em camadas típica: uma camada de unidades de entrada (e *bias*), conectada através de uma camada de pesos à camada com unidades de saída.

Computação de gradientes

Para treinar redes neurais como a vista acima por meio do método de descenso, precisamos ser capazes de computar o gradiente g da função erro com respeito a cada peso w_{ij} da rede. O gradiente nos diz como uma pequena variação nos pesos vai afetar o erro total E. Iniciamos com a quebra da função erro em termos separados para cada ponto p da base de treinamento:

$$E = \sum_p E^p \tag{2.5}$$

$$E^p = \frac{1}{2} \sum_o (t_o^p - y_o^p)^2 \tag{2.6}$$

onde o varia de acordo com as unidades de saída da rede, t_o^p é o valor desejado da o-ésima saída para o exemplo p, e y_o^p é a saída produzida pela rede. Já que diferenciação

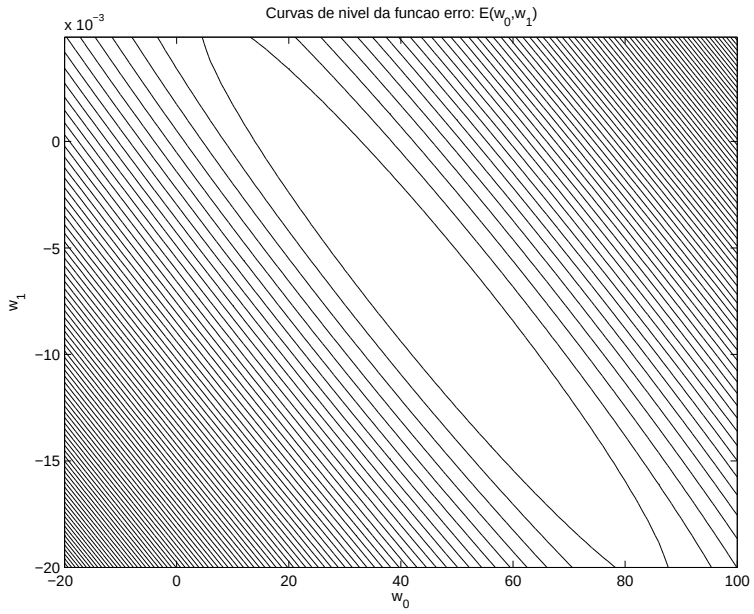

Figura 2.9: Curvas de nível do erro $E(w_0, w_1)$.

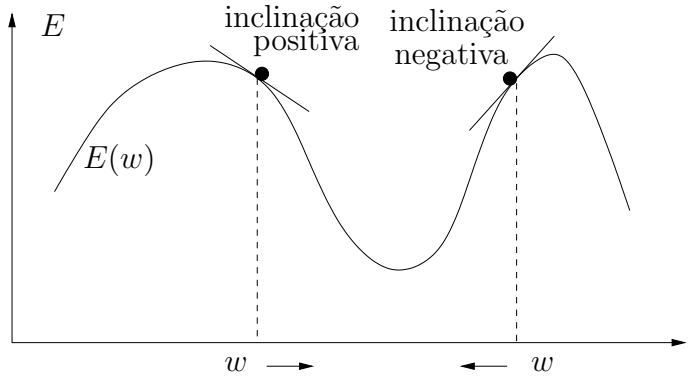

Figura 2.10: Direção para atualização de parâmetros conforme sinal do gradiente.

e somatória são intercambiáveis, podemos quebrar o gradiente em componentes separados para cada ponto de treinamento:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_p E^p = \sum_p \frac{\partial E^p}{\partial w_{ij}} \tag{2.7}$$

No que segue, descrevemos o cômputo do gradiente para um único ponto de treinamento, p.

Primeiro aplicamos a regra da cadeia para decompor o gradiente em dois fatores:

$$\frac{\partial E^p}{\partial w_{oi}} = \frac{\partial E^p}{\partial y_o^p} \frac{\partial y_o^p}{\partial w_{oi}} \tag{2.8}$$

O primeiro fator pode ser obtido diferenciando a equação (2.6) acima:

$$\frac{\partial E^p}{\partial y_o^p} = -(t_o^p - y_o^p) \tag{2.9}$$

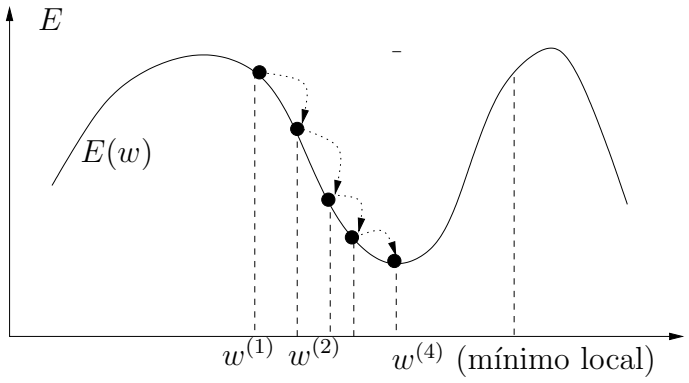

Figura 2.11: Progresso do método de descenso em direção ao mínimo local.

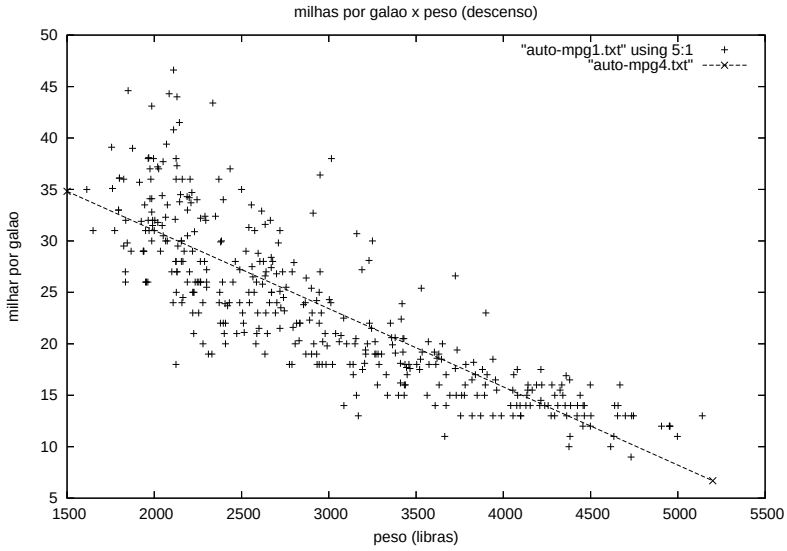

Figura 2.12: Aproximação linear da função milhas por galão obtida com o método de descenso. Utilizamos $\mu_0 = 10^{-4}$ como taxa de decrescimento para w_0 e $\mu_1 = 10^{-10}$ como taxa de decrescimento para w_1, pois as entradas e saídas não são da mesma ordem.

Uma vez que:

$$y_o^p = \sum_j w_{oj} y_j^p \tag{2.10}$$

podemos expressar o segundo termo de (2.8) como:

$$\frac{\partial y_o^p}{\partial w_{oi}} = \frac{\partial}{\partial w_{oi}} \sum_j w_{oj} y_j^p = y_i^p \tag{2.11}$$

Substituindo (2.9) e (2.11) em (2.8), obtemos:

$$\frac{\partial E^p}{\partial w_{oi}} = -(t_o^p - y_o^p) y_i^p \tag{2.12}$$

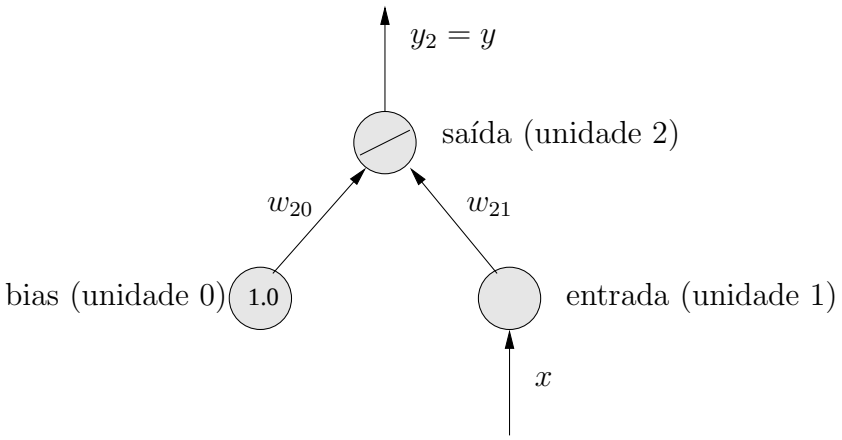

Figura 2.13: Rede neural.

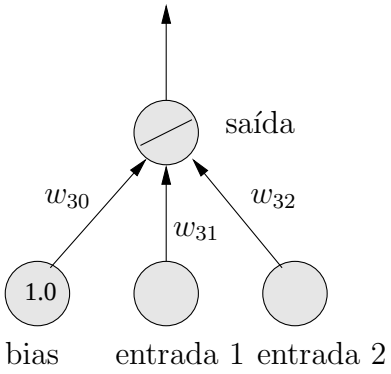

Figura 2.14: Rede neural linear simples com mais de uma entrada.

Para computar o gradiente ∇E do conjunto de pontos, basta somar a cada peso a contribuição dada pela equação (2.12) sobre todos os pontos de treinamento:

$$\frac{\partial E}{\partial w_{oi}} = \sum_p \frac{\partial E^p}{\partial w_{oi}} = -\sum_p (t_o^p - y_o^p) y_i^p \tag{2.13}$$

Note ainda que:

$$\nabla E(w) = \left[\frac{\partial E}{\partial w_{oi}} : \forall \text{saída } o, \text{entrada } i \right]$$

Podemos então subtrair uma proporção pequena μ, chamada de *"taxa de aprendizagem"*, de $\nabla E(w)$ dos pesos para executar uma iteração do método de descenso, ou seja, $\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k - \mu \nabla E(\mathbf{w}^k)$ com $\mathbf{w}^k$ correspondendo ao vetor de pesos no início da iteração k do algoritmo de descenso.

O algoritmo de descenso

1. Inicialize os pesos com valores pequenos aleatoriamente:
   $w_{ij} \in [-1, 1]$ aleatoriamente, para toda saída i e entrada j
2. Repita até convergência
3. Para cada peso w_{ij} defina $\Delta w_{ij} = 0$

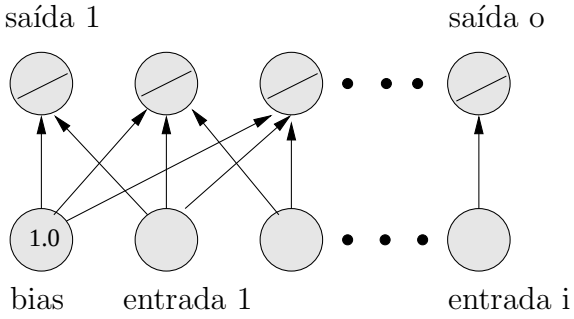

Figura 2.15: Rede neural linear para regressão múltipla.

Tabela 2.2: Sumário da regressão múltipla

	caso geral	rede linear
dados de treinamento	$(\mathbf{x}, \mathbf{t})$	$(\mathbf{x}, \mathbf{t})$
parâmetros do modelo	$\mathbf{w}$	$\mathbf{w}$
modelo	$\mathbf{y} = g(\mathbf{w}, \mathbf{x})$	$y_o = \sum_j w_{oj} y_j$
função erro	$E(\mathbf{y}, \mathbf{t})$	$E = \sum_p E^p, \quad E^p = \frac{1}{2}\sum_o (t_o^p - y_o^p)^2$
gradiente com respeito a w_{ij}	$\frac{\partial E}{\partial w_{ij}}$	$-\sum_p (t_i^p - y_i^p) y_j^p$
regra de atualização	$\Delta w_{ij} = -\mu \frac{\partial E}{\partial w_{ij}}$	$\Delta w_{ij} = -\mu \sum_p (t_i^p - y_i^p) y_j^p$

4. Para cada ponto de treinamento (x^p, t^p)
5. Coloque x^p nas entradas da rede
6. Calcule os valores das unidades de saída:
   Calcule y_i^p para cada unidade de saída i
7. Para cada peso w_{ij} defina:
   $\Delta w_{ij} = \Delta w_{ij} - (t_i^p - y_i^p) y_j^p$
8. Para cada peso w_{ij} defina:
   $w_{ij} = w_{ij} - \mu \Delta w_{ij}$

O algoritmo termina quando atingimos, ou estamos suficientemente próximos, de um mínimo da função erro, onde $\nabla E(w) = 0$. Neste ponto dizemos que o algoritmo convergiu.

A Tab. 2.2 faz uma analogia entre o método de descenso para o caso onde a rede neural é linear e o caso geral.

Taxa de aprendizagem

A taxa de aprendizagem μ determina o quanto se modifica os pesos w a cada passo. Se μ é muito pequeno, o algoritmo levará muito tempo para convergir (ver Fig. 2.16). Por outro lado, se μ é muito grande, o algoritmo pode entrar em laço infinito ou divergir (ver Fig. 2.17).

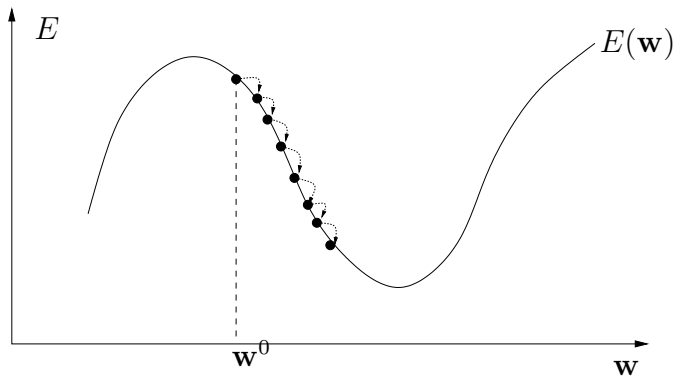

Figura 2.16: Taxa de aprendizagem pequena, implicando em convergência lenta.

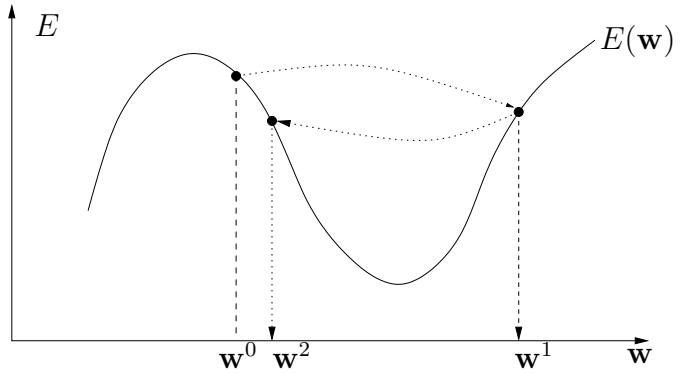

Figura 2.17: Taxa de aprendizagem grande, podendo levar a laço infinito e divergência.

Aprendizagem *"on-line"* versus *"batch"*

No desenvolvimento acima, as contribuições do gradiente de todos pontos de treinamento foram acumuladas antes de se atualizar os pesos. Este método é conhecido como aprendizagem *"batch"*. Um método alternativo é aprendizagem *"on-line"* onde os pesos são atualizados imediatamente após examinarmos cada ponto. Já que o gradiente de cada ponto de treinamento pode ser visto como uma aproximação com *"ruídos"* do gradiente total $\nabla E(w)$ (Fig. 2.18), o método também é conhecido como *"método de descenso estocástico"* (com ruído).

Aprendizagem *on-line* tem algumas vantagens:

1. tende a ser mais rápido, especialmente quando o conjunto de treinamento apresenta redundâncias (muitos pontos similares);

2. pode ser empregada quando não há um conjunto de treinamento fixo (dados chegam continuamente);

3. apresenta melhor desempenho no seguimento de ambientes não-estacionários (quando o problema é variante no tempo); e

4. o *"ruído"* no gradiente pode ajudar a escapar de mínimos locais (que são típicos e problemáticos nos casos não-lineares).

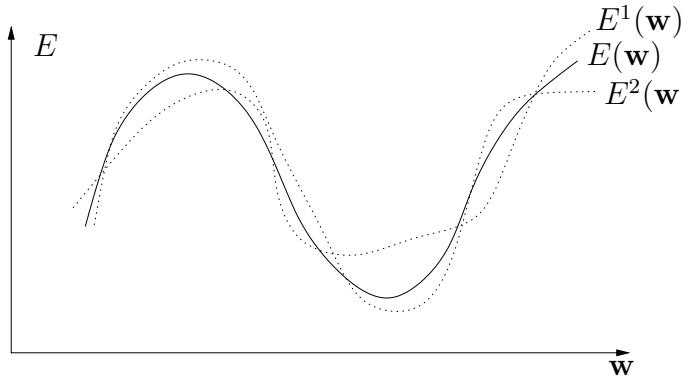

Figura 2.18: Aprendizagem *on-line*. $E(w)$ é a média de $E^p(w)$ sobre os pares de treinamento p.

Mas estas vantagens têm contra-partidas. Técnicas de otimização poderosas tais como métodos de gradiente conjugados, métodos de segunda-ordem e *"support vector machines"* são do tipo *"batch"*. Contudo a aplicação destes métodos demanda grande aprofundamento destas técnicas, o que pode ser relativamente complexo e consumir muito tempo.

Uma relação de compromisso pode ser obtida usando os métodos *"mini-batches"*: os pesos são atualizados após examinarmos m pontos de treinamento, onde m é maior que 1 mas menor que a cardinalidade do conjunto de treinamento. Por razões de simplicidade, o foco daqui a diante será em métodos *"on-line"*, dentre os quais o método de descenso é uma das melhores técnicas conhecidas. Aprendizagem *"on-line"* também é recomendável na implementação de estratégias de controle reativo em agentes adaptativos, o que deve ser aplicável em outras disciplinas relacionadas.

2.6 Redes multi-camadas

Um problema não-linear

Considere a melhor linha de regressão para os dados relativos a veículos. Observe que os pontos de treinamento não estão distribuídos de forma uniforme em torno da reta: para valores baixos, observamos milhas extras por galão em relação à predição. Parece que uma curva simples poderia levar a uma predição mais precisa do que uma linha. Podemos dotar a nossa rede neural para fazer o enquadramento de uma curva se um nó adicional apresentar uma função de ativação não-linear, que permita capturar a curvatura. Uma função útil para este propósito é a tangente hiperbólica com formato S (*tanh*) ilustrada na Fig. 2.19. Mais especificamente, a tangente hiperbólica pode ser expressa como:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

A Fig. 2.20 mostra a nova rede: um nó extra (unidade 2) com função de ativação *tanh* foi inserido entre a entrada e a saída. Já que tal nó é *"escondido"* dentro da rede, a unidade é dita *"unidade escondida"*. Veja que a unidade escondida tem

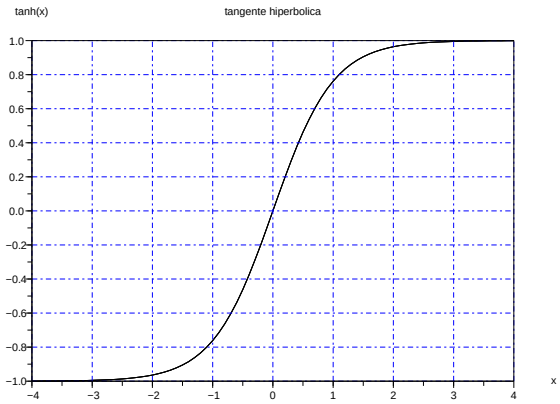

Figura 2.19: Tangente hiperbólica.

um peso para a unidade de *"bias"*. Em geral, todas as unidades que não sejam de entrada tem um peso associado ao *"bias"*. Para simplificar a apresentação, a unidade de *"bias"* e as unidades de entrada são omitidas dos diagramas de redes neurais — a menos que explicitamente mencionado, tais unidades estão presentes.

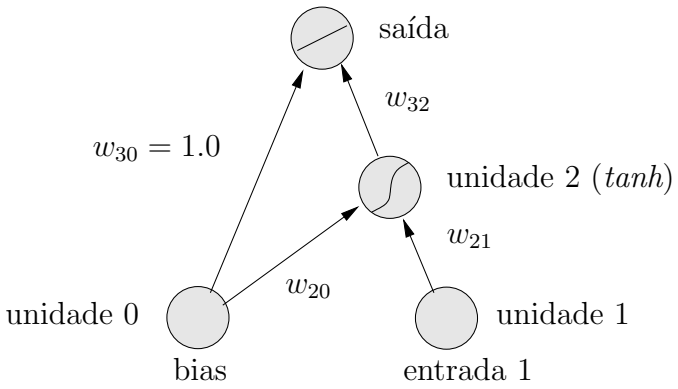

Figura 2.20: Rede neural com uma unidade escondida que implementa a função tangente hiperbólica *tanh*.

Quando a rede neural é treinada via método de descenso no caso de predição de milhas por galão em função do peso do veículo, ela aprende a enquadrar a função *tanh* aos dados. Cada um dos quatro pesos da rede desempenha um papel particular no processo: os dois pesos de *"bias"* deslocam a função *tanh* nas direções de x e y, respectivamente, enquanto os outros dois pesos distribuem a função ao longo das duas direções.

A rede da Fig. 2.20, obtida com parâmetros $w_{03} = 1.1$, $w_{20} = 1.1$, $w_{32} = -1.7$ e $w_{21} = -1.6$, produz uma aproximação da relação milhas por galão versus peso do veículo ilustrada na Fig. 2.21. Os dados da relação foram "*normalizados*" fazendo $w^{norm} \leftarrow w/1000 - 3$ e $mpg^{norm} \leftarrow mpg/10 - 2$, com w representando o peso do veículo e mpg representando milhas por galão.

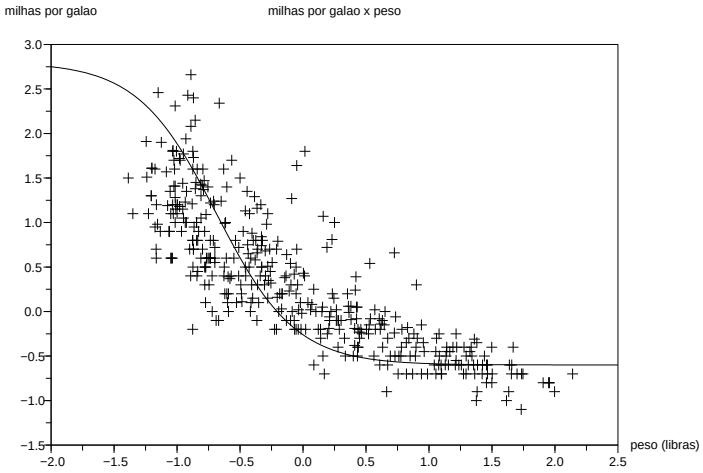

Figura 2.21: Aproximação da rede neural com uma unidade escondida (*tanh*) para a relação milhas por galão × peso do veículo (libras). Os dados da relação foram *"normalizados"* fazendo $w^{norm} \leftarrow w/1000 - 3$ e $mpg^{norm} \leftarrow mpg/10 - 2$.

Camadas escondidas

No exemplo acima selecionamos uma função de ativação para a unidade escondida capaz de enquadrar os dados. O que faríamos se os dados tivessem a forma dada na Fig. 2.22?

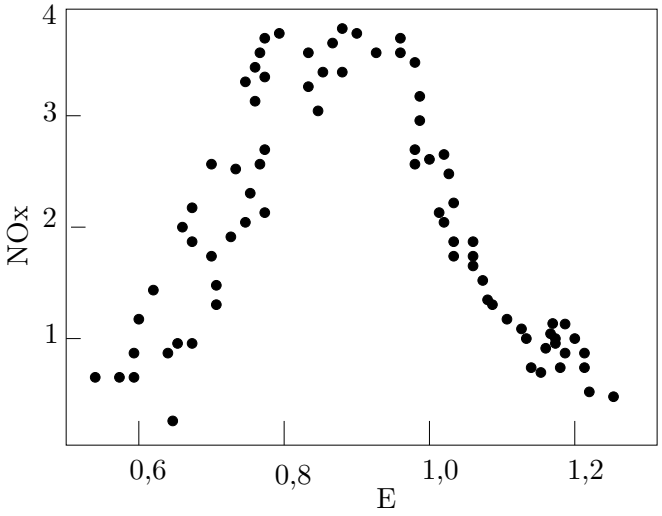

Figura 2.22: Concentração relativa de NO e NO_2 nos gases de exaustão como uma função da concentração de etanol/ar na mistura de combustão em um motor de carro.

Agora a função de ativação *tanh* não é mais capaz de capturar a relação não-linear dos dados. Em princípio, poderíamos conceber uma função de ativação para cada conjunto de dados encontrados, mas isto vai contra o propósito de aprendermos o modelo a partir dos dados. Gostaríamos de ter uma função geral e não-linear

capaz de aproximar qualquer conjunto de dados que nos for apresentado, qualquer que seja a sua forma. Há uma solução relativamente simples: basta adicionar mais unidades escondidas. Uma rede com apenas duas unidades escondidas com função de ativação tipo *tanh* (Fig. 2.23) é capaz de enquadrar os dados da Fig. 2.22 satisfatoriamente. A aproximação pode ser aprimorada adicionando uma terceira camada. No entanto, um número excessivo de unidades escondidas (ou camadas escondidas) pode degradar o desempenho da rede. Como linha geral, não devemos usar mais unidades escondidas do que o necessário para resolver um dado problema. Uma maneira de assegurarmos este princípio de projeto consiste em iniciarmos o treinamento com uma rede pequena. Se o método de descenso não encontrar uma solução satisfatória, então adicione unidades escondidas à rede e repita o processo de treinamento.

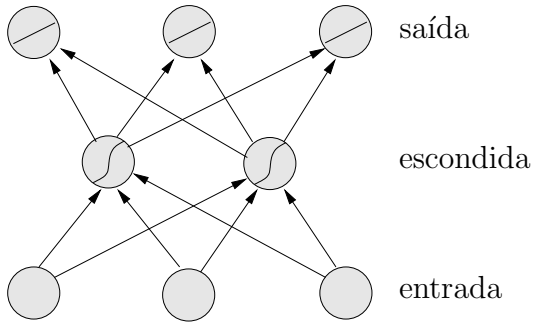

Figura 2.23: Rede neural com duas unidades escondidas do tipo tangente hiperbólica.

Resultados teóricos indicam que com um número suficiente de unidades escondidas, uma rede com estrutura conforme a Fig. 2.23 pode aproximar qualquer função com o grau de precisão desejado. Isto é, qualquer função pode ser expressa como uma combinação linear de funções *tanh* — *tanh* é uma base universal para funções. Outras funções podem constituir bases universais. As duas funções de ativação mais comumente utilizadas em redes neurais são as funções sigmoidais (em formato S) que incluem as funções *tanh* e as funções de base radial. A função *sigmóide* (σ) é definida como:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

2.7 Propagação reversa (*"backpropagation"*)

Já vimos o método de descendo para treinar redes lineares na Seção 2.4. Ao aplicarmos o mesmo método às redes multi-camadas encontramos uma dificuldade: não sabemos os valores desejados para as unidades escondidas. Isto parece um problema insolúvel — como poderíamos dizer o que as unidades escondidas devem fazer? Esta questão é uma das razões para as redes neurais terem sido descartadas, após um período inicial de grande popularidade, nos anos 50. Levaram cerca de 30 anos para que o algoritmo de propagação reversa (*"backpropagation"*), *"backprop"* de forma sintética, fosse desenvolvido para treinar as unidades intermediárias, instigando uma nova onda de pesquisas e aplicações em redes neurais.

"Backprop" provê o treinamento de redes com um número qualquer de unidades escondidas organizadas em um número qualquer de camadas. Há, todavia, limites práticos a serem vistos mais adiante. Na verdade, a rede não precisa ser organizada em camadas; qualquer padrão de conectividade que induz uma ordem parcial dos nós da entrada até a saída pode ser empregado. Ou seja, deve existir uma forma de ordenar as unidades tal que as conexões vão das mais recentes (próximas das entradas) em direção às mais tardias (próximas das saídas), o que equivale a dizer que os padrões de conexão não contêm ciclos. Redes que respeitam este padrão de conexões são conhecidas por *"feedforward networks"* — o padrão de conexão destas redes forma um grafo acíclico.

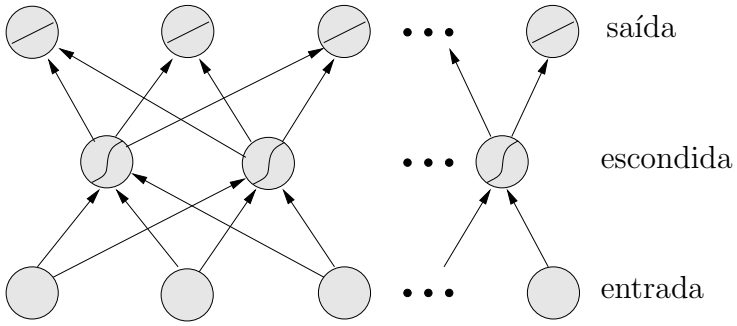

Figura 2.24: Rede neural com camada de unidades escondidas do tipo tangente hiperbólica.

Algoritmo *"backpropagation"*

Desejamos treinar uma rede multi-camada tipo *"feedforward"* com o algoritmo de descenso para aproximar uma função, tomando como base de treinamento pares $(\mathbf{x}, \mathbf{t})$. O vetor $\mathbf{x}$ representa o padrão de entrada para a rede, enquanto o vetor $\mathbf{t}$ corresponde ao objetivo (valor desejado para a saída). Como visto acima, o gradiente total para todo o conjunto de treinamento é a soma dos gradientes para cada padrão. Portanto, no que segue, vamos descrever o procedimento para cômputo do gradiente para apenas um padrão. Vamos numerar as unidades e denotar o peso de uma unidade j para uma unidade i por w_{ij}.

1. **Definições**
   - o sinal de erro para uma unidade j: $\delta_j = -\frac{\partial E}{\partial net_j}$
   - o gradiente (negativo) para o peso w_{ij}: $\Delta w_{ij} = -\frac{\partial E}{\partial w_{ij}}$
   - o conjunto de nós que precedem a unidade i: $A_i = \{j : \exists\ w_{ij}\}$
   - o conjunto de nós que seguem a unidade j: $P_j = \{i : \text{existe } w_{ij}\}$
2. **O gradiente.** Como fizemos em redes lineares, podemos expandir o gradiente em dois fatores por meio da regra da cadeia:

$$\Delta w_{ij} = -\frac{\partial E}{\partial net_i}\frac{\partial net_i}{\partial w_{ij}}$$

O primeiro fator é o erro da unidade i. O segundo fator é:

$$\frac{\partial net_i}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_{k \in A_i} y_k w_{ik} = y_j$$

Juntando as duas expressões, obtemos:

$$\Delta w_{ij} = \delta_i y_j$$

Para computar este gradiente, precisamos conhecer a atividade e o erro de todos os nós relevantes na rede.

3. **Propagação de ativação.**

   A atividade das unidades de entrada é determinada pela entrada externa $\mathbf{x}$. Para as demais unidades, a atividade é propagada à frente:

$$y_i = f_i(\sum_{j \in A_i} w_{ij} y_j)$$

   Note que antes de computarmos a atividade da unidade i, y_i, as atividades de suas unidades precedentes (as que formam o conjunto A_i) devem ser conhecidas. Já que as redes *"feedforward"* não contêm ciclos, há uma ordem das entradas em direção às saídas que respeita esta condição.

4. **Calculando o erro na saída.** Assumindo que estamos adotando a soma dos erros quadráticos como função erro:

$$E = \frac{1}{2} \sum_o (t_o - y_o)^2$$

   o erro para a unidade de saída o é simplesmente:

$$\delta_o = t_o - y_o$$

5. **Propagação reversa de erros.** Para as unidades escondidas, devemos propagar o erro para trás (daí surge o nome do algoritmo). Aplicando a regra da cadeia, podemos obter o erro de uma unidade escondida em termos dos seus nós posteriores:

$$\delta_j = -\sum_{i \in P_j} \frac{\partial E}{\partial net_i} \frac{\partial net_i}{\partial y_j} \frac{\partial y_j}{\partial net_j}$$

   Dos três fatores dentro da soma, o primeiro é simplesmente o erro do nó i. O segundo é:

$$\frac{\partial net_i}{\partial y_j} = \frac{\partial}{\partial y_j} \sum_{k \in A_i} w_{ik} y_k = w_{ij}$$

   Por outro lado a derivada da função de ativação do nó j nos dá:

$$\frac{\partial y_j}{\partial net_j} = \frac{\partial f_j(net_j)}{\partial net_j} = f_j'(net_j)$$

Para a unidade escondida h que usa tanh como função de ativação, podemos usar a identidade $tanh(u)' = 1 - tanh(u)^2$, que nos leva a:

$$f'_h(net_h) = 1 - y_h^2$$

Juntando todos os elementos obtemos:

$$\delta_j = f'_j(net_j) \sum_{i \in P_j} \delta_i w_{ij}$$

Para calcular o erro da unidade j, δ_j, devemos primeiramente saber o erro de todos os seus nós posteriores (os nós que formam P_j). Enquanto a rede não possuir ciclos, existe uma ordem dos nós da saída em direção à entrada que respeita esta condição. Tal ordem pode ser estabelecida revertendo o sentindo da ordem na qual as atividades eram propagadas à frente.

Forma matricial

Para redes tipo *"feedforward"* totalmente conectadas (onde cada nó em uma camada está conectado com cada nó da camada seguinte) há uma forma mais conveniente de descrever o algoritmo *"backprop"* em notação matricial. Os pesos de *"bias"*, as entradas líquidas, ativações e sinais de erro para todas as unidades de uma camada são combinados em vetores, enquanto todos os pesos que não surgem de *"bias"* entre uma camada e a próxima formam uma matriz W. As camadas são numeradas de 0 (camada de entrada) até L (camada de saída). O algoritmo *"backprop"* consiste dos passos abaixo:

1. **Inicialize a camada de entrada:**

$$\mathbf{y}_0 = \mathbf{x}$$

2. **Propague a atividade à frente:** para $l = 1, \ldots, L$,

$$\mathbf{y}_l = \mathbf{f}_l(W_l \mathbf{y}_{l-1} + \mathbf{b}_l)$$

onde $\mathbf{b}_l$ é o vetor de *"bias"* e $\mathbf{f}_l$ é uma função vetorial com a saída da camada l.

3. **Calcule o erro da camada de saída:**

$$\boldsymbol{\delta}_L = \mathbf{t} - \mathbf{y}_L$$

4. **Propague o erro para trás:** para $l = L-1, L-2, \ldots, 1$,

$$\boldsymbol{\delta}_l = (W_{l+1}^T \boldsymbol{\delta}_{l+1}).f'_l(\mathbf{net}_l)$$

onde T indica transposição de matriz.

5. **Atualize os pesos e** *"bias"***:**

$$\Delta W_l = \boldsymbol{\delta}_l \mathbf{y}_{l-1}^T$$
$$\Delta \mathbf{b}_l = \boldsymbol{\delta}_l$$

Esta notação é bem mais compacta do que a notação em grafos, apesar de descreverem exatamente a mesma seqüência de operações (considerando redes em camadas).

2.8 Classificação de padrões

Acabamos de desenvolver um método para treinamento de redes neurais. Dado um conjunto de entradas (x) e saídas desejas (y), a rede encontra um mapeamento de x para y. Dado um valor x que não faz parte da base de treinamento, a rede neural faz uma predição do valor de y. A habilidade de predizer (corretamente) a saída correspondente a uma entrada não vista é conhecida por *"capacidade de generalização"*.

Este estilo de aprendizagem é chamado de *"aprendizagem supervisionada"* (ou aprendizagem com tutor) pois nos são dados relações conhecidas entre entrada e saída. Aprendizagem não supervisionada busca encontrar padrões em dados em vez de encontrar um mapeamento entre entrada e saída. Vamos a seguir considerar outro tipo de aprendizagem supervisionada, além de regressão linear vista acima, aplicada ao problema de classificação de padrões.

Classificação de padrões

Um exemplo clássico de classificação de padrões é o reconhecimento de caracteres. A partir de conjuntos de valores de *"pixels"* associados à imagem de um caracter, procuramos um procedimento computacional que determine a letra de uma imagem ainda não classificada. Os valores dos *"pixels"* são as entradas e as categorias de letras correspondem às saídas.

A letra *"A"* por exemplo pode assumir uma forma bem diferente dependendo da fonte utilizada. No caso de escrita à mão, o padrão vai variar conforme a pessoa que escreve. Logo, existe uma região de valores para as variáveis de decisão que são mapeadas para a mesma classe. Isto é, se desenharmos os valores das variáveis de decisão, regiões distintas vão corresponder a classes distintas, conforme ilustra a Fig. 2.25.

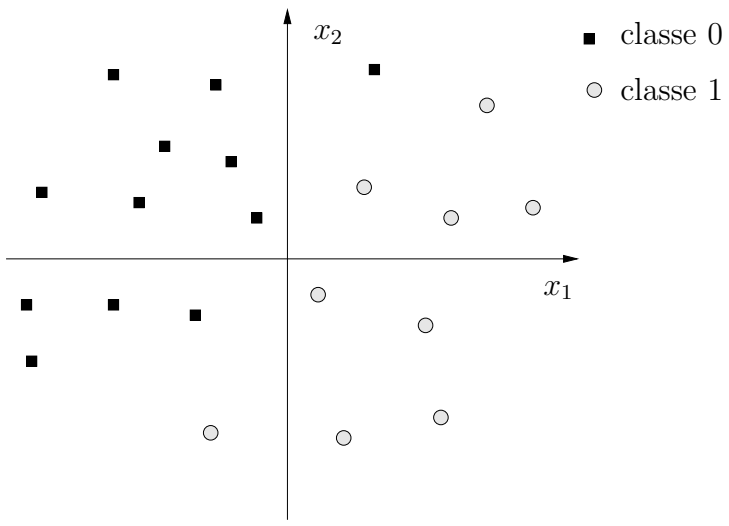

Figura 2.25: Exemplo de problema de classificação de padrões. Regiões são associadas aos padrões.

Classificação de padrões com redes de uma camada

Podemos seguir a abordagem de regressão linear onde os alvos passam a ser classes. Note que as saídas não são mais valores contínuos, mas assumem uma natureza discreta.

O que acontece se tivermos apenas duas classes? Com apenas duas classes, precisamos de apenas uma unidade de saída, de acordo com a ilustração da Fig. 2.26. O alvo é 1 se o exemplo está, digamos, na classe 1 e o alvo é 0 (ou -1) se o alvo está na classe 0. Parece razoável usar uma função degrau para garantir a saída apropriada.

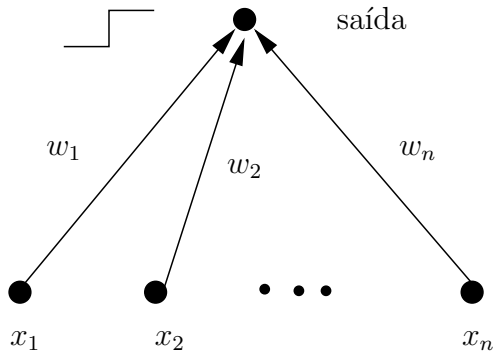

Figura 2.26: Rede neural com apenas uma unidade de saída para classificação em duas classes.

Vamos discutir dois métodos para treinamento de redes de uma camada para classificação de padrões:

- **Perceptron:** garante encontrar os pesos corretos caso existam.
- **Adaline:** (usa regra delta) pode ser facilmente generalizada para redes multicamada (problemas não-lineares).

Mas como podemos saber se existem pesos que resolvem o problema de classificação? Primeiramente vamos examinar o que a arquitetura de uma camada é capaz de representar.

Rede de uma camada com função degrau para ativação

Considere uma rede com dois nós de entradas e um nó de saída (2 classes). A saída líquida da rede é uma função linear dos pesos e das entradas:

$$net = \mathbf{w}^T\mathbf{x} = w_1x_2 + x_2w_2$$
$$y = f(net)$$

Suponha que desejamos que a saída assuma o valor 0 para a classe 0, e o valor 1 para a classe 1. Isto significa que desejamos:

- $net > 0$ para classe 1
- $net < 0$ para classe 0

A linha de divisão entre as duas classes é a linha definida por $net = 0$, ou seja, o conjunto de pontos que satisfazem a equação $w_1x_1 + w_2x_2 = 0$. Estas noções podem ser visualizadas na Fig. 2.27.

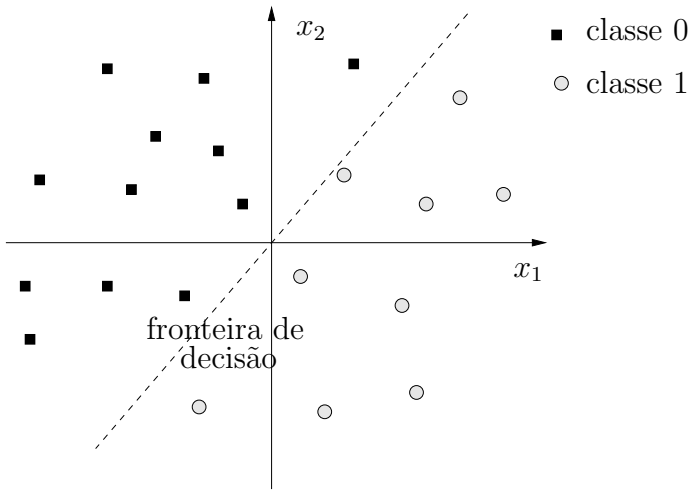

Figura 2.27: Exemplo de fronteira de decisão que passa pela origem.

O que aconteceria se a linha que divide as duas classes não passa pela origem? Ver Fig. 2.28 para um exemplo. Para descrever esta linha podemos introduzir uma constante b, para que a reta seja definida por $x_2 = ax_1 + b$. A constante b pode ser incorporada na rede através do *"bias"*. *Bias* apareceu inicialmente no caso de regressão e podemos fazer o mesmo nesta situação, adicionando uma entrada fictícia com valor fixo em 1. O peso associado com esta entrada é dito peso *bias*. Agora temos como equação:

$$\begin{aligned} net &= w_1x_1 + w_2x_2 + b \\ &= \mathbf{w}^T\mathbf{x} + b \\ &= 0 \end{aligned}$$

A rede neural aumentada do peso *"bias"* aparece na Fig. 2.29. Note que $f(net) = f(w_1x_1 + w_2x_2 + b) = 0$ se $net < 0$ e $f(net) = 1$ set $net > 0$.

Outras duas observações de interesse são:

1. o vetor de pesos $\mathbf{w} = (w_1, w_2)$ é normal à fronteira de decisão, ou seja, $\nabla_{\mathbf{x}} net = (\frac{\partial net}{\partial x_1}, \frac{\partial net}{\partial x_2}) = (w_1, w_2)$. Outra demonstração pode ser obtida via interpretação geométrica: seja $\mathbf{z}_2 - \mathbf{z}_1$ um vetor paralelo à fronteira de decisão; então, $\mathbf{z}_1, \mathbf{z}_2$ pertencem à fronteira, implicando $\mathbf{w}^T\mathbf{z}_1 + b = 0$ e $\mathbf{w}^T\mathbf{z}_2 + b = 0$ que, por sua vez, implica $\mathbf{w}^T(\mathbf{z}_2 - \mathbf{z}_1) = 0$; mas isto implica em $\mathbf{w}$ ser perpendicular a $\mathbf{z}_2 - \mathbf{z}_1$ e, portanto, perpendicular à fronteira de decisão.

2. a distância da origem à fronteira é $\frac{|b|}{\|\mathbf{w}\|}$. Já que $\mathbf{w}$ é ortogonal à fronteira, devemos encontrar um escalar c tal que $c\mathbf{w}$ esteja na fronteira, o que significa dizer que $\mathbf{w}^T(c\mathbf{w}) + b = 0 \implies c\|\mathbf{w}\|^2 = -b \implies c = -b/\|\mathbf{w}\|^2$; o comprimento do vetor $c\mathbf{w}$ é $\|c\mathbf{w}\| = |c|\|\mathbf{w}\| = \frac{|b|}{\|\mathbf{w}\|^2}\|\mathbf{w}\| = \frac{|b|}{\|\mathbf{w}\|}$.

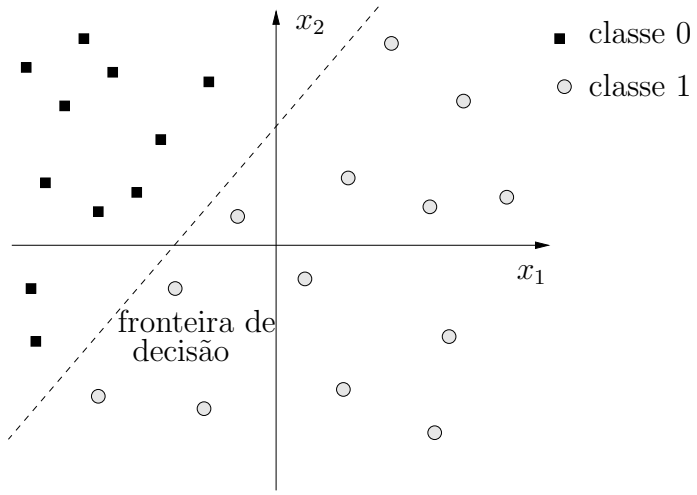

Figura 2.28: Exemplo de fronteira de decisão que não atravessa a origem.

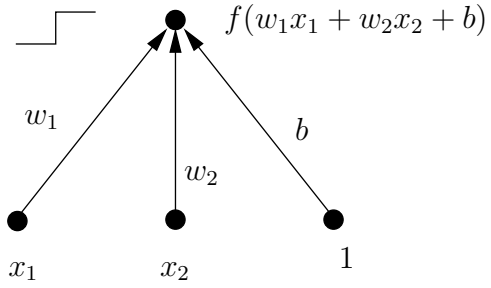

Figura 2.29: Rede neural, com ajuste de inclinação e interceptação, contendo apenas uma unidade de saída para classificação em duas classes.

Classes linearmente separáveis

Problemas de classificação cujas classes podem ser separadas por uma reta (hiperplano no caso multi-dimensional) são ditos problemas linearmente separáveis. Redes de uma camada só conseguem resolver problemas linearmente separáveis. Contudo, a maioria dos problemas encontrados na prática não são linearmente separáveis.

2.9 *Perceptron*

A regra de aprendizagem *"perceptron"* é um método para encontrar os pesos em uma rede. Considere o problema de aprendizagem supervisionada para classificação embora outros tipos de problemas possam também ser resolvidos. Uma propriedade desejável da regra de aprendizagem *"perceptron"* advém de sua habilidade de encontrar os pesos ótimos caso eles existam. Isto é verdade para representações binárias.

Hipóteses

- A rede possui apenas uma camada cuja saída é dada por:

$$output = f(net) = f(\mathbf{w}^T\mathbf{x})$$

  onde $\mathbf{w}$ é o vetor de pesos, $\mathbf{x}$ é o vetor de entradas e f é uma função degrau binária que assume os valores ± 1.

- O *"bias"* é tratado como uma outra entrada cujo valor é fixo em 1.

- m é o número de exemplos de treinamento $(\mathbf{x}, t)$ com t assumindo os valor $+1$ ou -1.

Interpretação geométrica

Com a função binária f, o problema se reduz a encontrar os pesos tal que $\text{sign}(\mathbf{w}^T\mathbf{x}) = t$. Ou seja, os pesos devem ser escolhidos de maneira que a projeção do padrão $\mathbf{x}$ em $\mathbf{w}$ tenha o mesmo sinal de t. Mas a fronteira entre projeções positivas e negativas é apenas o plano $\{\mathbf{x} : \mathbf{w}^T\mathbf{x} = 0\}$, ou seja, uma fronteira de decisão conforme a Fig. 2.30.

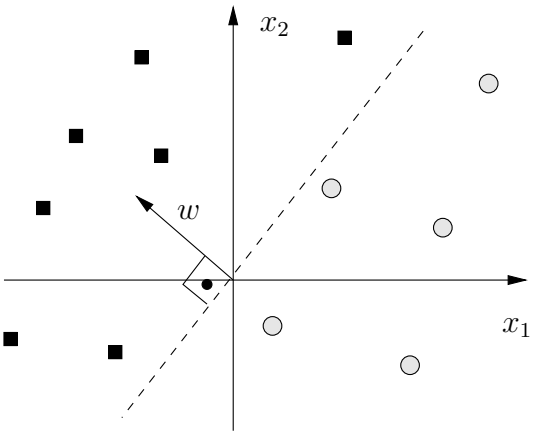

Figura 2.30: Classificação linear como um problema de projeção de $\mathbf{x}$ em $\mathbf{w}$.

Algoritmo *perceptron*

1. Inicialize os pesos (com valores pequenos aleatoriamente):
   $w_{ij} \in [-1, 1]$ aleatoriamente, para toda saída i e entrada j
2. Escolha uma taxa de aprendizagem α
3. Repita até convergência
4. Para cada ponto de treinamento $(\mathbf{x}^p, t^p)$
5. Coloque $\mathbf{x}^p$ nas entradas da rede
6. Calcule a ativação da unidade de saída:
   $y^p = f(\mathbf{w}^T\mathbf{x}^p)$
7. Se $y^p = t^p$, não modifique os pesos
8. Se $y^p \neq t^p$, atualize os pesos como segue:
9. $\mathbf{w}^{new} = \mathbf{w} + 2\alpha t^p\mathbf{x}^p$

Considere o que acontece quando ocorre treinamento dos padrões p_1 e p_2 indicados na Fig. 2.31. Note que os exemplos $\mathbf{x}^{p_1}$ e $\mathbf{x}^{p_2}$ estão incorretamente classificados pela fronteira de decisão (linha tracejada). Suponha que o exemplo p_1 é selecionado pelo algoritmo. Uma vez que $t^{p_1} = 1$, o vetor de pesos $\mathbf{w}$ é deslocado na direção de $\mathbf{x}^{p_1}$ dada pelo vetor $2\alpha t\mathbf{x}^{p_1}$.

Suponha que tivéssemos escolhido o padrão p_2 para executar uma iteração do algoritmo. O padrão p_2 tem classificação $t^{p_2} = -1$, logo o peso é movido de um vetor na direção $-\mathbf{x}^{p_2}$ de um comprimento proporcional dado por 2α, como indicado na Fig. 2.32. Em ambos os casos a nova fronteira leva à classificação correta dos padrões.

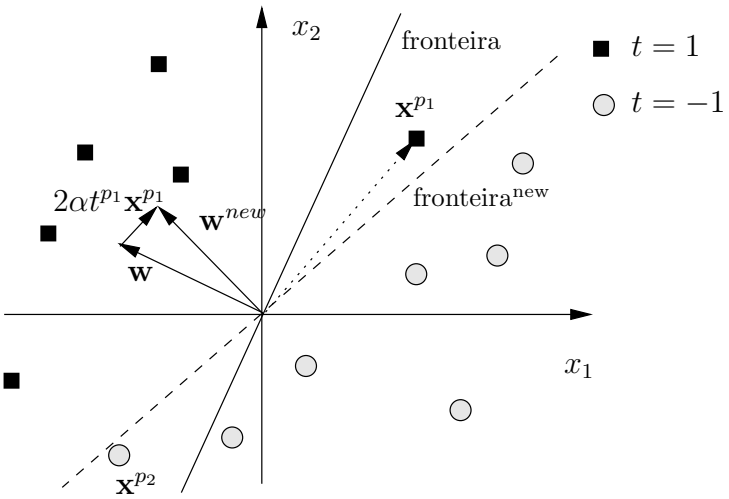

Figura 2.31: Ilustração do algoritmo de treinamento *"perceptron"*. Note que o vetor $2\alpha t\mathbf{x}^{p_1}$ é paralelo ao vetor $\mathbf{x}^{p_1}$, tendo a mesma direção e um comprimento dado por 2α vezes o comprimento de $\mathbf{x}^{p_1}$.

Comentários sobre *"perceptron"*

- A escolha da taxa de aprendizagem α não importa no resultado final, pois α apenas modifica a escala de $\mathbf{w}$.

- A fronteira de decisão (para o caso de duas entradas e um peso *bias*) tem equação:

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_3}{w_2} \tag{2.14}$$

  onde $w_3 = b$ representa o *"bias"*. A partir da equação (2.14) podemos perceber que a fronteira de decisão depende apenas de $\mathbf{w} = (w_1, w_2, w_3) = (w_1, w_2, b)$, mas não depende da taxa de aprendizagem.

- O algoritmo *"perceptron"* tem garantia de convergência em um número finito de passos se o problema é linearmente separável. Contudo, se o problema de classificação não for separável, o algoritmo pode se tornar instável.

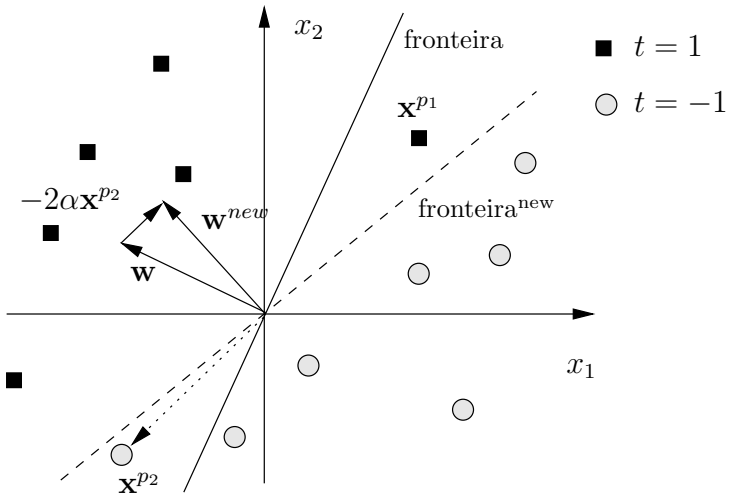

Figura 2.32: Ilustração do algoritmo de treinamento *"perceptron"*. Note que o vetor $2\alpha t^{p_2}\mathbf{x}^{p_2} = -2\alpha\mathbf{x}^{p_2}$ é paralelo ao vetor $\mathbf{x}^{p_2}$, mas tendo a direção oposta e um comprimento dado por 2α vezes o comprimento de $\mathbf{x}^{p_2}$.

2.10 Regra delta

A regra delta é também conhecida por outros nomes, tais como regra *"adaline"*, regra de Widrow-Hoff e mínimos quadrados. O princípio desta regra está em substituir a função degrau binária da regra *"perceptron"* por uma função de ativação contínua e diferenciável, tal como a função linear. No caso de problemas de classificação, a função degrau binária é aplicada apenas para determinar a classe dos exemplos, não sendo usada no cômputo dos pesos. Note que este algoritmo é, em essência, idêntico ao algoritmo de regressão linear visto acima, diferindo apenas na forma de determinar as classes. Esta diferença entre as duas regras de treinamento pode ser entendida através da Fig. 2.33.

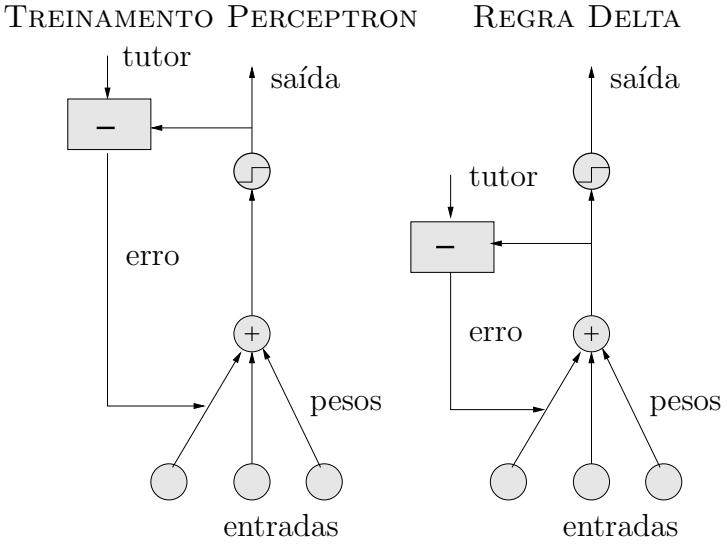

Figura 2.33: Diferença entre as arquiteturas dos algoritmos de treinamento com regra delta e *"perceptron"*.

Regra delta: método de descenso revisitado

Primeiramente, obtenha uma função custo E que afira o quão bem a rede aprendeu a classificar os padrões. Por exemplo, para o caso com uma saída única, podemos utilizar:

$$E = \frac{1}{2}\sum_{i=1}^{n}(t_i - y_i)^2$$

onde:

- n é o número de exemplos;
- t_i é o valor desejado associado ao i-ésimo exemplo; e
- y_i é a saída da rede obtida quando as entradas são alimentadas com o i-ésimo padrão.

Para treinar a rede, ajustamos os pesos no sentido de reduzir o custo, mas sob a hipótese de diferenciabilidade da função custo. Os passos principais do algoritmo são:

1. Inicialize os pesos com valores pequenos e aleatórios, por exemplo, valores uniformemente distribuídos no intervalo $[-1, 1]$.
2. Até que a função E atinja a tolerância desejada, atualize os pesos conforme as equações abaixo:

$$E = \frac{1}{2}\sum_{i=1}^{n}(t_i - y_i)^2 \tag{2.15}$$

$$\frac{\partial E}{\partial \mathbf{w}} = \sum_{i=1}^{n}(t_i - y_i)x_i \tag{2.16}$$

$$\mathbf{w}^{new} = \mathbf{w}^{old} - \alpha\frac{\partial E}{\partial \mathbf{w}} \tag{2.17}$$

onde E é avaliada para $\mathbf{w}^{old}$, α é a taxa de aprendizagem e $\frac{\partial E}{\partial \mathbf{w}}$ é o gradiente.

Trabalhando com mais de duas classes

Se existir mais de duas classes podemos utilizar a mesma rede, mas no lugar do valor de saída binário passamos a utilizar valores discretos. Por exemplo, para o caso de 5 classes, poderíamos utilizar os valores desejados como sendo $t \in \{1, 2, 3, 4, 5\}$ ou $t \in \{-2, -1, 0, +1, +2\}$. Contudo, a experiência teórico-prática nos diz que é mais adequado utilizar várias funções de saída. Em outras palavras, cada nó de saída passa a resolver um problema binário que corresponde à decisão de pertencer ou não a uma certa classe. A Fig. 2.34 dá uma idéia de como fronteiras de decisão individualizadas poderiam ser utilizadas para discriminar a pertinência a uma classe específica. A rede neural que implementa este classificador aparece na Fig. 2.35.

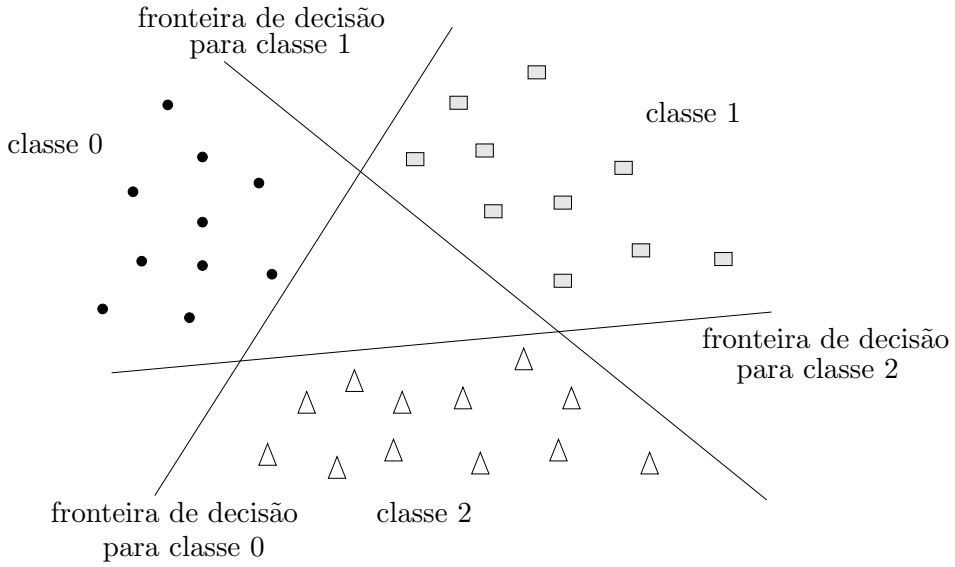

Figura 2.34: Exemplo de problema de classificação com múltiplas classes.

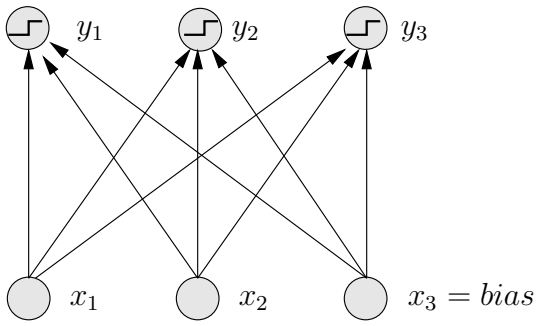

Figura 2.35: Rede neural que implementa classificador de múltiplas classes.

2.11 Princípios para projeto de classificadores

Aqui discutimos princípios gerais para concepção de redes neurais empregadas em problemas de classificação.

2.11.1 Método antigo

No caso de existirem mais de duas classes, foi sugerida a seguinte conduta:

- associe uma saída a cada uma das classes;
- defina o valor alvo de saída 1 quando o padrão pertencer a uma certa classe, e o valor 0 caso contrário;
- projete uma rede linear com função erro igual a soma dos erros quadráticos; e
- determine a classe de predição para um padrão como a saída de maior valor.

Há dois problemas com este método. Primeiro, a definição da função erro e a determinação das classes são desconexas. Segundo, um erro mínimo não necessariamente

produz a rede com o maior número de predições corretas. Contudo, é possível fazer alguns ajustes neste método e contornar estas inconsistências. Iniciaremos com uma mudança da interpretação da saída.

2.11.2 Método revisado

Segundo a nova interpretação, y_i passa a ser a probabilidade do padrão pertencer à classe i. Para tanto:

- cada saída deve estar entre 0 e 1; e
- a soma das saídas para cada padrão deve igualar a 1.

Como podemos implementar tais restrições? Há vários pontos a serem considerados.

Podemos substituir a função de ativação linear por outra função como, por exemplo, a função *"sigmoide"*[1]. *"Sigmoides"* variam continuamente entre 0 e 1, satisfazendo assim a primeira restrição aplicada sobre probabilidades. (A função *"sigmoide"* é uma boa escolha?)

Podemos também variar a função custo. Não estamos restritos a utilizar o erro quadrático médio EQM (do inglês, *mean squared error*). *Quais são as outras opções?* Para responder esta questão podemos pensar sobre o que faz sentido intuitivamente. Aplicando o método de descenso sobre a função EQM de uma rede linear, descobrimos que os pesos eram proporcionais ao erro ($Erro^p = (t^p - y^p)$) Isso faz sentido. Se passarmos a utilizar a função *"sigmoide"* como função de ativação obteremos uma fórmula mais complicada:

$$\frac{\partial E}{\partial w} = (t - y)y(1 - y)$$

Esta forma não exatamente o que procuramos. Contudo, há uma combinação de função erro e função de ativação que produz os resultados esperados (entropia cruzada e *"softmax"*, respectivamente).

Função erro (entropia cruzada)

A função erro adequada para a tarefa que estamos tratando é a *"entropia cruzada"* (do inglês, *"cross entropy"*) definida por:

$$E^p = \sum_{j=1}^{c} t_j^p \ln(y_j^p) \tag{2.18}$$

onde c é o número de classes (i.e., número de nós de saída da rede) e p é um padrão a ser treinado. A equação (2.18) tem suas raízes na teoria da informação, sendo aplicada com freqüência quando as saídas (y) são interpretadas como probabilidades. Não vamos nos aprofundar nas suas origens, porém tentaremos verificar se ela faz sentido em alguns casos especiais. Suponha que a rede é treinada com perfeição a mapear com exatidão os padrões às suas classes.

[1] A função *"sigmoide"* é definida por $\sigma(x) = 1/(1 + e^{-ax})$ onde a é uma constante positiva. Também é conhecida por função logística. Quanto maior a constante a, mais inclinada será a curva.

Suponha que a classe 3 é escolhida. Isto significa que a saída do nó 3 deve ser 1 (a probabilidade da classe 3 corresponder ao padrão é máxima) enquanto a saída dos nós das demais classes deve ser 0 (probabilidade mínima). Neste caso, o resultado da equação (2.18) é zero, como desejado.

Suponha que a rede dá saída $y_j = \frac{1}{2}$ para todas as saída indicando que há total incerteza sobre a classe do padrão. Neste caso, E assume o seu valor máximo.

Função de ativação (*"softmax"*)

A função de ativação *"softmax"* é definida por:

$$f_j(x) = \frac{e^{x_j}}{\sum_{k=1}^{c} e^{x_k}}, \quad j = 1, \ldots, c \tag{2.19}$$

onde f_j é a função de ativação da j-ésima saída e c é o número de classes. Observe que a função *"softmax"* tem boas propriedades:

- seu valor sempre está entre 0 e 1; e
- quando combinada com a função erro dá pesos proporcionais a $(t - y)$, como pode ser observado no desenvolvimento abaixo:

$$\begin{aligned}
E &= \sum_{j=1}^{c} t_j \ln(y_j) \\
&= \sum_{j=1}^{c} t_j \ln(f_j(net)) \\
\frac{\partial E}{\partial w_{rs}} &= \sum_{j=1}^{c} \frac{t_j}{f_j(net)} \frac{\partial f_j(net)}{\partial w_{rs}} \\
\frac{\partial f_j(net)}{\partial w_{rs}} &= \sum_{k=1}^{c} \frac{\partial f_j(net)}{\partial net_k} \frac{\partial net_k}{\partial w_{rs}} \\
&= t_j(\delta_{jr} - y_r)x_s \\
\frac{\partial E}{\partial w_{rs}} &= \sum_{j=1}^{c} t_j(\delta_{jr} - y_r)x_s
\end{aligned}$$

onde $\delta_{ij} = 1$ se $i = j$ e $\delta_{ij} = 0$ quando $i \neq j$. Note que se r é a classe correta então $t_r = 1$ e o lado-direito da equação acima se reduz a $(t_r - y_r)x_s$. Se $q \neq r$ é a classe correta, então $t_r = 0$ e a expressão acima também se reduz a $(t_r - y_r)x_s$. Logo temos:

$$\frac{\partial E}{\partial w_{rs}} = (t_r - y_r)x_s \tag{2.20}$$

A equação (2.20) é famliar? Note que ela tem a mesma forma da derivada do erro em relação a w da regra delta, equação (2.16), apresentada na Seção 2.10.

2.12 Referências

O conteúdo apresentado neste capítulo é uma compilação traduzida das notas de aula de Orr *et al.* [12] e do livro texto de Mitchell [10].

Uma ferramenta excelente para síntese de redes neurais, de vários tipos, e com algoritmos de treinamento otimizados é o *Matlab*. Informações sobre o *Toolbox* para redes neurais podem ser obtidas no site:

http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/nnet_product_page.html.

2.13 Exercícios

EX: Verifique se a entropia cruzada assume o valor máximo quando $y_j = \frac{1}{2}$ para toda a saída j.

Capítulo 3

Lógica Fuzzy (Nebulosa)

"Fuzzy logic fits best when variables are continuous and/or mathematical models do not exist." [Earl Cox, 1992]

3.1 Introdução

Um controlador fuzzy procura imitar as ações do operador através de conjuntos fuzzy. Para uma ilustração, considere o tanque de cimento em pó dado na Fig. 3.1, que alimenta o misturador a uma taxa mais ou menos constante. O projeto simplificado possui dois sensores e uma válvula magnética. O objetivo é controlar a válvula V_1 tal que o tanque seja reabastecido quando o nível ser torne baixo (LL), interrompendo o abastecimento quando o nível se torna alto (LH). O sensor LL emite o sinal 1 quando o nível está acima da marca, 0 quando o nível está abaixo da marca. O sensor LH opera de forma análoga. A válvula abre quando V_1 é colocado em 1 e fecha quando V_1 é colocado em 0. Na lógica Booleana, o controlador poderia ser descrito por:

$$V_1 = \begin{cases} 1, & \text{se } LL \text{ passa de 1 para 0} \\ 0, & \text{se } LH \text{ passa de 0 para 1} \end{cases} \tag{3.1}$$

Um operador cuja responsabilidade é abrir e fechar a válvula poderia seguir a estratégia:

$$\begin{cases} \text{Se o nível é baixo então abra } V_1 \\ \text{Se o nível é alto então feche } V_1 \end{cases} \tag{3.2}$$

A estratégia (3.1) é adequada para CLPs (controladores lógico-programáveis) que utilizam lógica Booleana, contudo a estratégia (3.2) é recomendada para um controlador baseado em lógica fuzzy. Aqui o objetivo não é dar detalhes da implementação do controlador fuzzy, mas fazer uso do exemplo para explicar os princípios da lógica fuzzy.

Lofti Zadeh, o pai da lógica fuzzy, sugere que muitos conjuntos encontrados no mundo não são definidos por uma fronteira clara. Por exemplo, o *"conjunto das montanhas altas"* ou o *"conjunto das medições baixas"* (conforme Fig. 3.1) são exemplos de tais conjuntos. Zadeh propôs a extensão da lógica binária, $\{0, 1\}$, para o domínio contínuo, intervalo $[0, 1]$, dessa forma permitindo uma transição gradual da

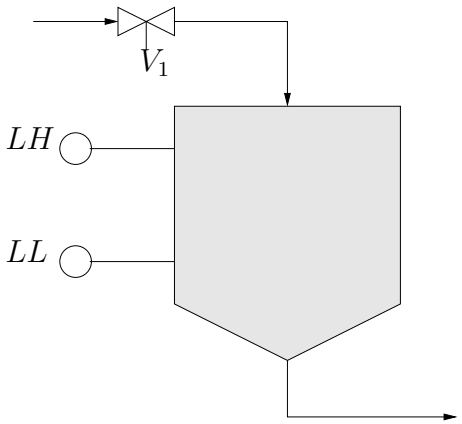

Figura 3.1: Tanque de alimentação de cimento em pó.

falsidade para a verdade. Os artigos originais sobre teoria fuzzy [17, 18, 19] mostram como a teoria de conjuntos fuzzy resulta da extensão da teoria de conjuntos.

A seguir, vamos nos concentrar na teoria fuzzy que toma coma base o sistema (3.2), apresentando as definições e operações básicas deste domínio.

3.2 Conjuntos fuzzy

Conjuntos fuzzy são desdobramentos da noção matemática de conjunto. Conjuntos foram estudados formalmente por Cantor (1845-1918). A teoria de conjuntos encontrou muita resistência no princípio, contudo é hoje apreciada pelos matemáticos pela sua capacidade de expressão. Muitos pesquisadores estão estudando as conseqüências da teoria de conjuntos fuzzy, com vasta literatura matemática. Para engenheiros de controle, lógica fuzzy e relações fuzzy são os conceitos mais importantes para se entender como que regras fuzzy funcionam.

Conjuntos convencionais

Um conjunto é qualquer coleção de objetos que podem ser tratados como um todo. Cantor descreveu um conjunto por meio de seus membros, tal que um item de um certo universo é membro ou não do conjunto. Os termos *"conjunto"*, *"coleção"* e *"classe"* são usados como sinônimos, da mesma forma que *"item"*, *"elemento"* e *"membro"* se referem à mesma entidade. Quase tudo que é dito conjunto em conversa corriqueira pode ser um conjunto aceitável em matemática.

Exemplos de conjuntos: As listas ou coleções de objetos abaixo são conjuntos:

- O conjunto dos inteiros não-negativos inferiores a 4. Este é um conjunto finito com quatro elementos: $\{0, 1, 2, 3\}$.

- O conjunto dos dinossauros que vivem no campus da Universidade Federal de Santa Catarina. Este é um conjunto vazio.

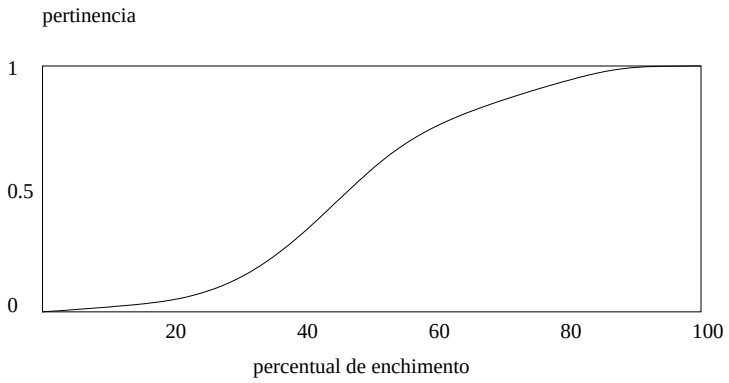

Figura 3.2: Grau de pertinência no caso do tanque de cimento em pó.

- O conjunto de medidas maiores que 10 volts. Apesar deste conjunto ser infinito, é possível determinar se uma dada medida é um membro ou não.

Um conjunto pode ser especificado por seus elementos — os elementos dão uma caracterização completa. A lista de membros $A = \{0, 1, 2, 3\}$ define um conjunto finito. Não é possível listar todos os elementos de um conjunto infinito. Devemos encontrar uma propriedade que caracteriza todos os elementos do conjunto, por exemplo, todos os números reais $x > 10$. Logo há duas maneiras de descrever um conjunto: *"explicitamente"* por meio de uma lista ou *"implicitamente"* por meio de um predicado que deve ser satisfeito pelos membros.

Conjuntos fuzzy

De acordo com Zadeh, conjuntos podem ter mais de um critério de pertinência além de estar contido ou não. Tome como exemplo o *"conjunto das pessoas jovens"*. Um bebê com um ano de idade certamente pertence a este conjunto, e uma pessoa de 100 anos certamente não está presente no conjunto, mas o que podemos dizer sobre as pessoas de 20, 30 e 40 anos de idade? Outro exemplo advém do noticiário meteorológico com altas temperaturas, fortes ventos ou dias belos. Em outros casos o critério parece não ser nebuloso, mas é entendido como nebuloso: a velocidade limite de 60 km/h, o horário de *"check-out"* do hotel, um homem com 50 anos de idade. Zadeh propôs um *"grau de pertinência"*, de forma que a transição entre pertinência e não-pertinência é gradual e não abruta.

O grau de pertinência para todos os seus membros descreve um conjunto fuzzy. O grau de pertinência de um elemento é um número entre 0 e 1, freqüentemente denotado pela letra grega μ. Quanto mais alto o número, maior o grau de pertinência (Fig. 3.2). De acordo com Zadeh o conjunto de Cantor é um caso especial onde os elementos têm pertinência completa, ou seja, $\mu = 1$. Mesmo assim chamaremos os conjuntos com pertinência completa de conjuntos não-fuzzy de Cantor. Observe que Zadeh não dá uma base formal para determinar o grau de pertinência. O grau de pertinência de uma pessoa de 50 anos ao conjunto de pessoas jovens depende da visão de cada um. O grau de pertinência é uma noção precisa mas subjetiva que depende do contexto.

O grau de pertinência fuzzy difere da noção estatística de distribuição de probabilidade. Isto pode ser ilustrado no exemplo abaixo:

Exemplo (probabilidade × possibilidade): (Zadeh in [20]) Considere a afirmação *"José comeu X ovos durante o café da manhã"*, onde $X \in U = \{1, 2, \ldots, 8\}$. Podemos associar a distribuição de probabilidades p observando como *José* se alimenta de ovos no café da manhã por cerca de 100 dias:

$$\begin{array}{rcl} U & = & [\quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad] \\ p & = & [\ 0,1 \ \ 0,8 \ \ 0,1 \ \ 0 \quad 0 \quad 0 \quad 0 \quad 0 \ ] \end{array}$$

Um conjunto fuzzy, que expressa o grau de possibilidade de *José* ter comido X ovos, pode ser dado pela distribuição de possibilidade π:

$$\begin{array}{rcl} U & = & [\ 1 \ \ 2 \ \ 3 \ \ 4 \quad 5 \quad\ 6 \quad\ 7 \quad\ 8 \ ] \\ \pi & = & [\ 1 \ \ 1 \ \ 1 \ \ 1 \ \ 0,8 \ \ 0,6 \ \ 0,4 \ \ 0,2 \ ] \end{array}$$

onde a possibilidade de $X = 3$ é 1, enquanto a probabilidade é apenas $0,1$.

Discussão: *O exemplo mostra que um evento possível não implica que ele é provável. Por outro lado, se um evento é provável então ele deve ser possível. Podemos ver a função de pertinência fuzzy como uma distribuição pessoal, em contraste com a distribuição estatística que é baseada em observações.*

Universo

Elementos de um conjunto fuzzy são tomados a partir de um *"universo"*. O universo contém todos os elementos que podem ser considerados. Até mesmo o universo depende do contexto, como ilustra o exemplo abaixo.

Exemplo (universo):

a) O conjunto das pessoas jovens poderia ter todas as pessoas do mundo como universo. Por outro lado, o universo poderia ser a faixa de 0 a 100, que representariam a idade como mostra a Fig. 3.3.

b) O conjunto dos $x > 10$ (x deve ser maior do que 10) poderia ter como universo todas as medições positivas.

O emprego do universo visa suprimir o uso de dados incorretos, por exemplo medições negativas do nível do tanque. Quando lidamos com quantidade não numéricas como, por exemplo, *"sabor"* que não pode ser medido em uma escala numérica, não podemos fazer uso de um universo numérico. Os elementos são tomados a partir de um conjunto de noções psicológicas: por exemplo, o universo poderia ser {amargo, doce, azedo, salgado, quente, ...}.

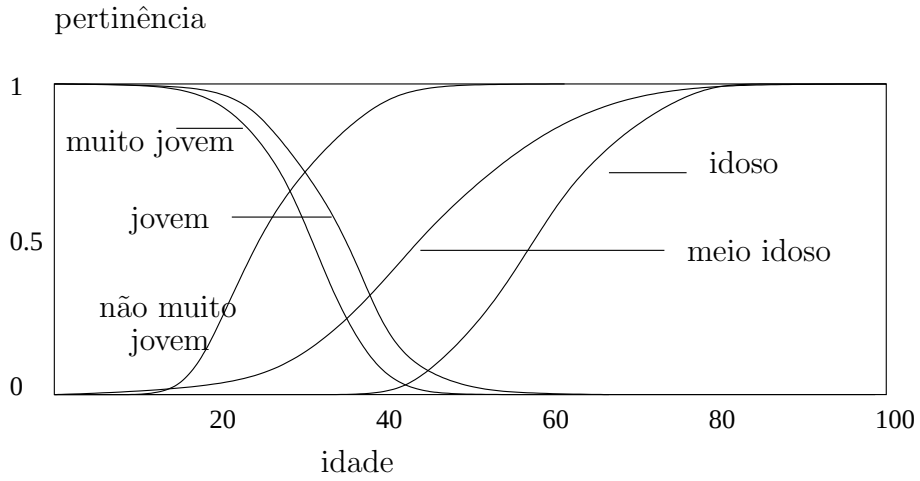

Figura 3.3: Função pertinência para o conjunto fuzzy de pessoas jovens.

Funções de pertinência

Cada elemento do universo tem um grau de pertinência ao conjunto fuzzy, talvez nulo. O conjunto de elementos que têm grau de pertinência não nulo é dito *"conjunto suporte"* do conjunto fuzzy. A função que associa um número a cada elemento x do universo é dita função de pertinência, sendo denotada por $\mu(x)$.

Representações contínuas e discretas: Há duas alternativas para representar funções de pertinência em um computador: contínua ou discreta. No caso contínuo, a função de pertinência é uma função matemática, possivelmente um programa. Exemplos de funções de pertinência são a curva π, a curva s, a curva z, a triangular e a trapezoidal. A Fig. 3.3 apresenta um exemplo de função s. No caso discreto, a função de pertinência e o universo são pontos de uma lista (vetor). Algumas vezes pode ser conveniente representá-la por meio de amostras.

Em general, a forma contínua demanda mais CPU, mas requer menos memória dos que a forma discreta.

Exemplo (função de pertinência contínua): A função coseno pode ser usada para gerar uma variedade de funções de pertinência. A curva s pode ser gerada fazendo:

$$s(x_l, x_r, s) = \begin{cases} 0, & x < x_l \\ \frac{1}{2} + \frac{1}{2}\cos(\frac{x-x_r}{x_r-x_l}\pi), & x_l \leqslant x \leqslant x_r \\ 1, & x > x_r \end{cases}$$

onde x_l é o ponto de quebra à esquerda e x_r é o ponto de quebra à direita.

A curva z é apenas uma reflexão da curva s:

$$z(x_l, x_r, s) = \begin{cases} 1, & x < x_l \\ \frac{1}{2} + \frac{1}{2}\cos(\frac{x-x_l}{x_r-x_l}\pi), & x_l \leqslant x \leqslant x_r \\ 0, & x > x_r \end{cases}$$

A curva π pode ser implementada por meio da combinação das curvas s e z, tal

que o pico seja constante no intervalo $[x_2, x_3]$:

$$\pi(x_1, x_2, x_3, x_4, x) = \min(s(x_1, x_2, x), z(x_3, x_4, x))$$

Exemplo (função de pertinência discreta): Para uma representação discreta da Fig. 3.2, suponha que o universo é representado pelas amostras:

$$u = [0\ 20\ 40\ 60\ 80\ 100]$$

Podemos estabelecer a relação de pertinência através da lista de valores:

$$\begin{aligned}
&\pi(10, 90, 100, 100, u_1) = 0\\
&\pi(10, 90, 100, 100, u_2) = 0,04\\
&\pi(10, 90, 100, 100, u_3) = 0,31\\
&\pi(10, 90, 100, 100, u_4) = 0,69\\
&\pi(10, 90, 100, 100, u_5) = 0,96\\
&\pi(10, 90, 100, 100, u_6) = 1
\end{aligned}$$

Normalização: Um conjunto fuzzy é *"normalizado"* se o maior valor de pertinência é 1. Um conjunto pode ser normalizado dividindo o valor de pertinência de cada elemento pelo maior valor de pertinência: $a/\max(a)$.

Pares fuzzy

Um conjunto fuzzy A é uma coleção de pares:

$$A = \{(x, \mu(x))\}$$

O item x pertence ao universo e $\mu(x)$ é o grau de pertinência a A. Um par $(x, \mu(x))$ é dito *"par fuzzy"*, logo o conjunto pode ser visto como a união de *"pares fuzzy"*. Pode ser conveniente pensar em A como um vetor:

$$a = (\mu(x_1), \mu(x_2), \ldots, \mu(x_n))$$

onde se entende que cada posição i $(1, 2, \ldots, n)$ corresponde a um ponto do universo.

Variáveis linguísticas

Da mesma forma que variáveis algébricas assumem valores numéricos, uma *"variável lingüística"* pode ter palavras ou sentenças como valores. O conjunto de valores que ela pode assumir é dito *"conjunto de termos"*. Cada valor no conjunto de termos é uma *"variável fuzzy"* definida sobre a *"variável base"*. A *"variável base"* define o universo do discurso para todos as variáveis fuzzy no conjunto de termos. Em outras palavras, a hierarquia é dada por: variável lingüística → variável fuzzy → variável base.

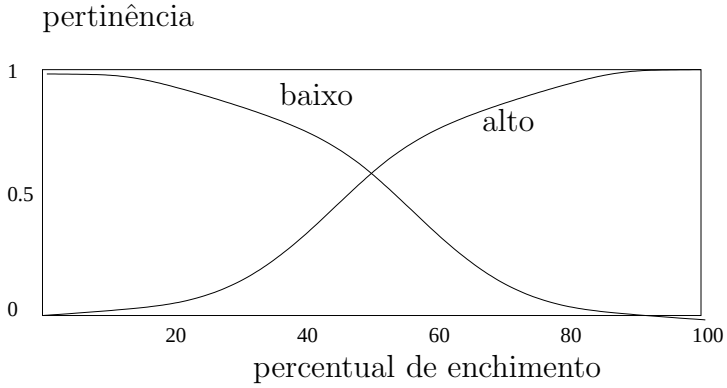

Figura 3.4: Conjunto de termos {*baixo,alto*} para o problema do tanque de cimento.

Exemplo (conjunto de termos): Seja x uma variável lingüística sobre idade. Termos desta variável lingüística, que são conjuntos fuzzy, podem ser: *"idoso"*, *"jovem"* e *"muito jovem"*. O conjunto de termos pode ser colocado como:

$$T = \{\textit{idoso, não tão velho, meio jovem, jovem, muito jovem}\}$$

Cada termo do conjunto de termos T é uma variável fuzzy sobre a variável base, a qual pode ter como escala valores entre 0 e 100 anos.

Exemplo do tanque

Agora estamos mais próximos da representação de uma regra de controle fuzzy. Na premissa,

Se o *nível* **é** *baixo*, ...

claramente *"baixo"* é uma variável fuzzy, um valor da variável lingüística *"nível"*. Está definida sobre um universo, que é a faixa de valores esperados para *"nível"*, ou seja, o intervalo $[0, 100]$ com percentuais de tanque cheio. As medidas de *"nível"* são escalares, e a declaração *"nível* **é** *baixo"* corresponde ao valor de pertinência *"nível*(i)*"*, onde *"nível"* é arredondado para o elemento mais próximo do universo para encontrar um nível apropriado para i. O saída é um número $\mu \in [0, 1]$ que diz quão bem a premissa é satisfeita. A Fig. 3.4 sugere uma definição para o conjunto de termos {*baixo,alto*} para o problema do tanque.

3.3 Operações sobre conjuntos fuzzy

A função de pertinência é obviamente um componente crucial de um conjunto fuzzy. É então natural definir operações sobre conjuntos fuzzy através de suas funções de pertinência.

Operações sobre conjuntos

Operações sobre conjuntos fuzzy criam um novo conjunto fuzzy, ou vários conjuntos, como ilustra a Fig. 3.5, onde os conjuntos fuzzy $A \cap B$, $A \cup B$ e $\overline{A \cup B}$ são criados a partir dos conjuntos A e B.

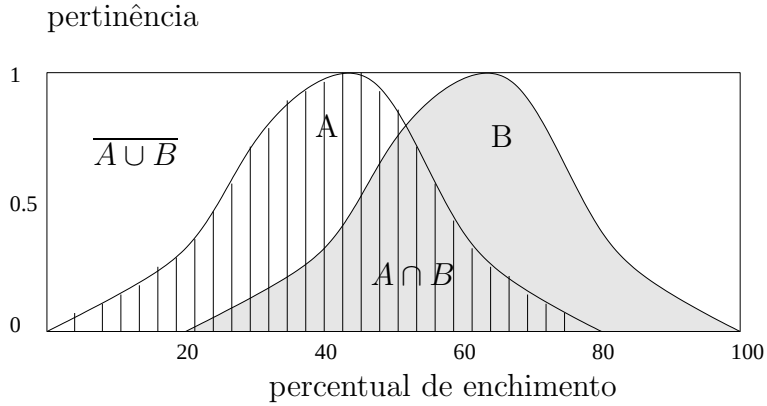

Figura 3.5: Operações sobre conjuntos fuzzy.

Definição 1 *Sejam A e B dois conjuntos fuzzy em um mesmo universo. A interseção de A e B é definida por:*

$$A \cap B \equiv a \min b$$

Ou seja o grau de pertinência de um elemento x em relação a $A \cap B$ é dado por $\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x))$

Definição 2 *A união de dois conjuntos fuzzy é dada por:*

$$A \cup B \equiv a \max b$$

Ou seja, ou grau de pertinência de um elemento x ao conjunto $A \cup B$ é dado por $\mu_{A \cup B} = \max(\mu_A(x), \mu_B(x))$.

Definição 3 *O conjunto fuzzy completo de A ($\bar{A}$) é definido por:*

$$\bar{A} = 1 - a$$

Ou seja, o grau de pertinência de um elemento x em relação ao conjunto $\bar{A}$ é dado por $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$.

Definição 4 *Um conjunto fuzzy X é um subconjunto fuzzy de Y, escrito $X \subseteq Y$, se sua função de pertinência for menor ou igual à função de pertinência de Y para todos os elementos do universo comum. Ou seja, $\mu_X(x) \leqslant \mu_Y(x)$. Na Fig. 3.5 temos que $(A \cap B) \subseteq (A \cup B)$*

Exemplo (comprando uma casa): Uma família com quatro integrantes deseja comprar uma casa. Uma indicação de conforto se refere ao número de quartos de dormir. Eles também desejam comprar uma casa grande. Seja $u = (1, 2, \ldots, 10)$ o conjunto de casas disponíveis descritas pelo número de quartos de dormir. Então o conjunto fuzzy c que caracteriza conforto pode ser descrito como:

$$c = [0,2 \;\; 0,5 \;\; 0,8 \;\; 1 \;\; 0,7 \;\; 0,3 \;\; 0 \;\; 0 \;\; 0 \;\; 0]$$

Tabela 3.1: Operações sobre conjuntos fuzzy

Propriedade	Nome
$A \cup B = B \cup A$	comutatividade
$A \cap B = B \cap A$	comutatividade
$(A \cup B) \cup C = A \cup (B \cup C)$	associatividade
$(A \cap B) \cap C = A \cap (B \cap C)$	associatividade
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	distributividade
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	distributividade
$\overline{A \cap B} = \overline{A} \cup \overline{B}$	DeMorgan
$\overline{A \cup B} = \overline{A} \cap \overline{B}$	DeMorgan
$(A \cap B) \cup A = A$	Absorção
$(A \cup B) \cap A = A$	Absorção
$A \cup A = A$	Idempotência
$A \cap A = A$	Idempotência
$A \cup \overline{A} \neq 1$	Exclusão não é satisfeita
$A \cap \overline{A} \neq 0$	Exclusão não é satisfeita

Seja i o conjunto fuzzy caracterizando a noção de grande. O conjunto i pode ser caracterizado por:

$$i = [0 \;\; 0 \;\; 0,2 \;\; 0,4 \;\; 0,6 \;\; 0,8 \;\; 1 \;\; 1 \;\; 1 \;\; 1]$$

A interseção entre confortável e grande é dado por:

$$c \cap i = [0 \;\; 0 \;\; 0,2 \;\; 0,4 \;\; 0,6 \;\; 0,3 \;\; 0 \;\; 0 \;\; 0 \;\; 0]$$

Interpretando o conjunto fuzzy $c \cap i$, concluímos que uma casa com 5 quartos é ótima, mas satisfatória com grau 0,6. A segunda melhor solução é a casa com 4 quartos.

A união de confortável e grande nos dá:

$$c \cup i = [0,2 \;\; 0,5 \;\; 0,8 \;\; 1 \;\; 0,7 \;\; 0,8 \;\; 1 \;\; 1 \;\; 1 \;\; 1]$$

Agora uma casa com 4 quartos é totalmente satisfatória porque é confortável, e as casas com 7 a 10 quartos também porque são grandes.

O complemento de grande produz:

$$\overline{i} = [1 \;\; 1 \;\; 0,8 \;\; 0,6 \;\; 0,4 \;\; 0,2 \;\; 0 \;\; 0 \;\; 0 \;\; 0]$$

As operações $\cup$ e $\cap$ são associativas e comutativas. Estas propriedades são importantes, pois nos ajudam a predizer o resultado de sentenças longas. A Tab. 3.1 apresenta outras propriedades.

Modificadores

Um modificador lingüístico é uma operação que modifica o significado de um termo. Por exemplo, na sentença *"muito próximo de 0"*, a palavra *"muito"* modifica

a sentença *"próximo de 0"* que é um conjunto fuzzy. Um modificador é portanto uma operação sobre um conjunto fuzzy. Exemplos de outros modificadores são *"pouco"*, *"mais ou menos"*, *"possivelmente"* e *"com certeza"*. Embora seja difícil deixar preciso o significado do efeito do modificador *"muito"*, com certeza ele produz um efeito intensificador. O modificador *"mais ou menos"* tem o efeito oposto. Eles são muitas vezes aproximados pelas operações:

$$muito \equiv (a \rightarrow a^2)$$
$$mais\ ou\ menos \equiv (a \rightarrow \sqrt{a})$$

No caso de conjuntos discretos, considere o universo $u = (0, 20, 40, 60, 80)$. Dado o conjunto:

$$jovem = [1 \quad 0,6 \quad 0,1 \quad 0 \quad 0]$$

Então podemos derivar a função de pertinência para o conjunto *"muito jovem"* elevando todos os termos ao quadro, o que produz:

$$muito\ jovem = jovem^2 = [1 \quad 0,36 \quad 0,01 \quad 0 \quad 0]$$

Da mesma forma, o conjunto *"muito muito jovem"* é obtido fazendo:

$$muito\ muito\ jovem = jovem^4 = [1 \quad 0,13 \quad 0 \quad 0 \quad 0]$$

Outros exemplos de modificadores são:

$$extremamente \equiv (a \rightarrow a^3)$$
$$ligeiramente \equiv (a \rightarrow a^{\frac{1}{3}})$$

Uma família de modificadores pode ser gerada fazendo a^p onde p é a potência entre zero e infinito. Quando $p = \infty$ o modificador pode ser dito *"exatamente"*, pois isto força a nulificação de pertinência de todas as entradas menores do que 1.

Relações entre conjuntos

Em um controlador fuzzy, relações entre objetos desempenham um papel relevante. Algumas relações se referem a elementos de um mesmo universo: uma medida é maior que outra, um evento ocorreu antes de outro, ou um elemento se assemelha a outro. Outras relações se referem a elementos de universos disjuntos: a medida é grande e a taxa de variação é positiva, o valor de x é grande enquanto o de y é pequeno. Estes são exemplos de relações entre dois objetos, mas em princípio podemos ter relações entre um número qualquer de objetos.

Formalmente, uma *"relação binária"* ou simplesmente *"relação R"* de um conjunto A para um conjunto B designa para cada par $(a, b) \in A \times B$ precisamente uma das sentenças:

i) a está relacionado a b; ou

ii) a não está relacionado a b.

O produto cartesiano $A \times B$ é o conjunto de todas as possibilidades de combinarmos itens de A com itens de B. Uma *"relação fuzzy"* de um conjunto A para um conjunto B é um subconjunto fuzzy do produto cartesiano $U \times V$ entre os respectivos universos U e V.

Como exemplo, suponha que *José* tem um sobrinho *João* que se parece com o sobrinho *Pedro* com um grau de 0,8, mas *João* também se assemelha ao sobrinho *Marcos* com um grau de 0,9. Temos assim uma relação de semelhança entre os sobrinhos de *José*, convenientemente representada pela matriz:

$$R_1 = \begin{array}{c} \\ João \end{array} \begin{array}{cc} Pedro & Marcos \\ 0,8 & 0,9 \end{array}$$

Para ilustramos a composição de duas relações, considere a relação entre o tio *José* e os sobrinhos *Pedro* e *Marcos*:

$$R_2 = \begin{array}{c} \\ Pedro \\ Marcos \end{array} \begin{array}{c} José \\ 0,5 \\ 0,6 \end{array}$$

Podemos tentar descobrir o quanto *João* se parece com o tio *José* combinando as informações nas relações R_1 e R_2:

i) *João* se parece com *Pedro* com um grau de 0,8, enquanto *Pedro* se parece com o tio *José* com um grau de 0,5; ou

ii) *João* se parece com *Marcos* com um grau de 0,9, enquanto *Marcos* se parece com o tio *José* com um grau de 0,6.

A declaração (i) consiste de uma cadeia de relações e parece razoável encadeá-la através da operação de interseção. Isto corresponde a escolher o valor de pertinência menor para a relação transitiva *João* $\rightarrow$ *José*, ou seja, 0,5. Podemos proceder de maneira similar na situação (ii). Realizando a operação nas cadeias (i) e (ii) obtemos:

iii) *João* se parece com o tio *José* com um grau de 0,5; ou

iv) *João* se parece com o tio *José* com um grau de 0,6.

As declarações (iii) e (iv) são igualmente válidas, então é razoável aplicarmos o operador de união. Isto corresponde a escolhermos a relação mais forte, ou seja, o valor máximo das relações. O resultado final nos dá:

v) *João* se parece com o tio *José* com um grau de 0,6.

A regra geral para composição de relações fuzzy consiste em tomarmos o mínimo em uma série de conexões, e tomarmos o máximo em conexões paralelas. É conveniente realizarmos isto por meio do *"produto interno"*.

O produto interno é similar ao produto interno de matrizes (*do inglês*, *"dot product"*), exceto que a multiplicação é substituída pela interseção ($\cap$) e a soma é substituída pela união ($\cup$). Suponha que R é uma matriz $m \times p$ e S é uma matriz

$p \times n$. Então o produto interno $\cup \cdot \cap$ é uma matriz $m \times n = T = [t_{ij}]$ onde a entrada ij é obtida combinando a i-ésima linha de R com a j-ésima coluna de S, tal que:

$$\begin{aligned} t_{ij} &= (r_{i1} \cap s_{1j}) \cup (r_{i2} \cap s_{2j}) \cup \ldots (r_{ip} \cap s_{pj}) \\ &= \bigcup_{k=1}^{p} (r_{ik} \cap s_{kj}) \end{aligned}$$

De acordo com as definições adotadas para interseção e união (*min* e *max*), a composição de duas relações se reduz ao que é conhecido na literatura por *"composição max-min'*.

Se R é uma relação de a para b e S é uma relação de b para c, então a composição de R e S é uma relação de a para c (lei transitiva).

Exemplo (produto interno): Para as tabelas R_1 e R_2 dadas acima, temos:

$$\begin{aligned} R_1 \cup \cdot \cap R_2 &= [0,8 \quad 0,9] \cup \cdot \cap \begin{bmatrix} 0,5 \\ 0,6 \end{bmatrix} \\ &= \bigcup [0,5 \quad 0,6] \\ &= 0,6 \end{aligned}$$

3.4 Lógica fuzzy

Lógica teve como início o estudo da linguagem de argumentação, podendo ser aplicada para julgar corretude de uma cadeia de raciocínio em demonstrações matemáticas, por exemplo. Na lógica binária as proposições são *"verdadeiras"* ou *"falsas"*, mas não ambas. A *"veracidade"* ou *"falsidade"* designada a uma sentença é o *"valor da sentença"*. Por outro lado, na *"lógica fuzzy"* a proposição pode ser verdadeira, falsa ou ter um valor intermediário entre verdade e falsidade, tal como *"talvez verdadeira"*. A sentença *"o nível está alto"* é um exemplo de uma proposição que pode ser encontrada em um controlador fuzzy. Pode ser conveniente restringir os valores verdadeiros a um domínio discreto, digamos $\{0,\ 0,5,\ 1\}$ para *"falso"*, *"talvez verdadeiro"* e *"verdadeiro"* — neste caso estamos lidando com lógica de múltiplos valores. Na prática, uma divisão mais fina pode ser necessária e mais apropriada.

Conectivos

Em conversação cotidiana e matemática, sentenças são conectadas por meio de palavras *"e"*, *"ou"*, *"se-então"* (ou *"implica"*) e *"se-e-somente-se"* Estas palavras são ditas *"conectivos"*. Uma sentença modificada pela palavra *"não"* é dita *"negação"* da sentença original. A palavra *"e"* é usada para juntar duas sentenças formando uma *"conjunção"* de duas sentenças. De maneira similar a sentença formada ao conectarmos duas sentenças com a palavra *"ou"* é dita *"disjunção"* das duas sentenças.

A partir de duas sentenças podemos construir a forma *"se ... então ..."* que é dita sentença *"condicional"*. A sentença que segue o *"se"* é o *antecedente*, enquanto a sentença que segue o *"então"* é o *"conseqüente"*. Outros idiomas que podemos

Tabela 3.2: Tabela verdade para $p \vee q$ (forma usual ou Cartesiana)

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

considerar como tendo o mesmo significado de *"se p, então q"* (onde p e q são sentenças) são *"p implica q"*, *"p apenas se q"* e *"q se p"*.

As palavras *"se e somente se"* são usadas para obter uma sentença *"bicondicional"* a partir de duas sentenças.

Por meio de letras gregas e símbolos especiais, os conectivos pode ser estruturados e apresentados de forma mais eficaz. A notação tipicamente adotada é:

$\neg$ para *"não"*
$\wedge$ para *"e"*
$\vee$ para *"ou"*
$\Rightarrow$ para *"implica"*
$\Leftrightarrow$ para *"se e somente se"*

Exemplo (basquete): Considere a sentença:

> **Se** os *"piratas"* **ou** os *"leões"* perderem **e** os *"gigantes"* ganharem, **então** os *"corredores"* perderão a primeira colocação **e** eu deixarei a artilharia.

A sentença é um condicional simbolizado por $r \Rightarrow s$. O antecedente é composto de três sentenças: p (os piratas perderam), c (os leões perderam) e g (os gigantes ganharam). O conseqüente é a conjunção da sentença d (os corredores perderão a primeira colocação) e b (eu deixarei a artilharia). A sentença pode então ser simbolizada por:

$$((p \vee c) \wedge g) \Rightarrow (d \wedge b) \tag{3.3}$$

Os valores verdadeiro-falso possíveis para uma sentença podem ser sumarizados em uma tabela verdade. Tomemos como exemplo a tabela verdade da proposição $p \vee q$. A forma usual (Cartesiana) lista todas as combinações possíveis e os respectivos valores, conforme Tab. 3.2. A forma de Cayley coloca os argumentos p e q em linhas e colunas, como mostra a Tab. 3.3. O eixo vertical corresponde ao primeiro argumento (p), já o eixo horizontal fica com o segundo argumento (q). Na interseção da linha i com a coluna j temos o valor da expressão $p_i \vee q_j$. Os valores nos eixos para a forma de Cayley podem ser omitidos, no caso de lógica de dois valores, já que estes são sempre 0 e 1, e nesta ordem. Tabelas verdade para conectivos binários podem, portanto, ser representadas por matrizes 2×2, na qual se entende que o primeiro argumento está associado com o eixo vertical e o segundo, como o eixo horizontal. Um total de 16 tabelas deste tipo podem ser construídas, cada uma delas associada com um conectivo.

Tabela 3.3: Tabela verdade para $p \vee q$ (forma de Cayley)

$\vee$	0	1	q
0	0	1	
1	1	1	
p			

Tabela 3.4: Tabela verdade $((p \vee c) \wedge g) \Rightarrow (d \wedge b)$ (há 10 possibilidades de vitória dentre 32 possibilidades)

p	c	g	d	b
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
1	0	0	0	0
1	0	0	1	0
1	1	0	0	0
1	1	0	1	0

É possível avaliar, pelo menos em princípio, uma sentença lógica pelo teste exaustivo de todas as combinações de valores das variáveis da tabela verdade, o chamado *"arranjo lógico"*. O próximo exemplo ilustra o procedimento de *"arranjo lógico"*.

Exemplo (arranjo lógico): No exemplo de basquete, tínhamos $((p \vee c) \wedge g) \Rightarrow (d \wedge b)$. A sentença contém cinco variáveis e cada uma delas pode assumir dois valores. Isto significa que temos $2^5 = 32$ combinações possíveis. Apenas 23 delas são combinações legais, no sentido de que as sentenças são válidas (verdadeiras) para tais combinações, portanto, $32 - 23 = 9$ casos são ilegais (ou seja, a sentença é inválida para tais combinações de valores). Assumindo que estamos interessados apenas nas combinações onde me mantenho na artilharia ($b = 0$), então a Tab. 3.4 nos dá os valores verdade.

Tabelas semelhantes podem ser construídas na lógica fuzzy. Iniciamos definindo tabelas verdade para as operações de negação e disjunção, já que podemos derivar as demais operações a partir destas. Assumiremos que a *"negação"* é definida como o completo, ou seja, *not* $p \equiv 1 - p$ induz a função de pertinência $\mu_{\neg p}(x) = 1 - \mu_p(x)$. A *"disjunção"* é equivalente à operação de união teórica, ou seja, $p \vee q \equiv p \max q$ que induz a função de pertinência $\mu_{p \vee q}(x) = \max(\mu_p(x), \mu_q(x))$. Podemos então encontrar tabelas verdade para as operações *"ou"*, *"não-ou"*, *"não-e"* e *"e"*. A Tab. 3.5 mostra o resultado da operação *"ou"*. Note que $\mu_p(x), \mu_q(x) \in \{0,\ 0,5,\ 1\}$ indicando os casos que x não pertence ao conjunto fuzzy, talvez pertença e pertence ao conjunto fuzzy. Outras operações são ilustradas na Tabelas 3.6, 3.7 e 3.8.

Note que a Tabela 3.8 é a *"negação"* da Tabela 3.7. Similarmente, a Tabela 3.5 é

Tabela 3.5: Tabela verdade da operação *"ou"*: $p \vee q$

0	0,5	1
0,5	0,5	1
1	1	1

Tabela 3.6: Tabela verdade da operação *"não-ou"*: $\neg(p \vee q)$

1	0,5	0
0,5	0,5	0
0	0	0

Tabela 3.7: Tabela verdade da operação *"não-e"*: $(\neg p) \vee (\neg q)$

1	1	1
1	0,5	0,5
1	0,5	0

Tabela 3.8: Tabela verdade da operação *"e"*: $\neg((\neg p) \vee (\neg q))$

0	0	0
0	0,5	0,5
0	0,5	1

Tabela 3.9: Tabela verdade da operação *"implicação de Gödel"*: $p \Rightarrow q \equiv (p \leqslant q) \vee q$

1	1	1
0	1	1
0	0,5	1

Tabela 3.10: Tabela verdade da operação *"equivalência"*: $(p \Rightarrow q) \wedge (q \Rightarrow p)$

1	0	0
0	1	0,5
0	0,5	1

a *"negação"* da Tabela 3.6. Apesar da tabela verdade do *"e"* poder ser derivada das tabelas do *"ou"* e do *"não"*, podemos obter a tabela do *"e"* por meio da operação *min*, o que está de acordo com a definição do conjunto interseção.

O operador de *"implicação"*, por outro lado, trouxe complicações aos teóricos de lógica fuzzy. Se definirmos o operador na forma usual, ou seja, $p \Rightarrow q \equiv \neg p \vee q$, obteremos uma tabela verdade que é contra-intuitiva e inadequada porque várias leis lógicas deixam de ser respeitadas.

Vários pesquisadores tentaram desenvolver definições alternativas, chegando a uma lista com mais de 72 possibilidades. Uma outra possibilidade é a *"implicação de Gödel"* que é mais adequada pois mais relações da lógica clássica são preservadas (lógica de dois valores, verdadeiro ou falso). Três exemplos são:

1. $(p \wedge q) \Rightarrow p$ (simplificação)
2. $[p \wedge (p \Rightarrow q)] \Rightarrow q]$ (*"modus ponens"*)
3. $[(p \Rightarrow q) \wedge (q \Rightarrow r)] \Rightarrow (p \Rightarrow r)$ (silogismo hipotético)

A implicação de Gödel pode ser escrita como:

$$p \Rightarrow q \equiv (p \leqslant q) \vee q \tag{3.4}$$

A tabela verdade para equivalência ($\Leftrightarrow$) pode ser determinada a partir da implicação e conjunção, se estivermos de acordo que $p \Leftrightarrow q$ é o mesmo que $(p \Rightarrow q) \wedge (q \Rightarrow p)$. A tabela verdade da implicação pode ser visualizada na Tab. 3.9, enquanto a tabela verdade da equivalência aparece na Tab. 3.10.

Exemplo (*"modus ponens fuzzy"***):** É possível provar uma lei enumerando todas as combinações de valores das variáveis em lógica fuzzy, assumindo que o domínio das variáveis é discreto e limitado. Tome como exemplo o *modus ponens*:

$$[p \wedge (p \Rightarrow q)] \Rightarrow q$$

A sentença tem duas variáveis e assumiremos que cada variável pode tomar, digamos, três valores. Isto implica que temos $3^2 = 9$ combinações que estão ilustradas na Tab.

Tabela 3.11: Tabela verdade do *"modus ponens"*: $[p \wedge (p \Rightarrow q)] \Rightarrow q$

p	q	$p \Rightarrow q$	$[p \wedge (p \Rightarrow q)]$	$[p \wedge (p \Rightarrow q)] \Rightarrow q$
0	0	1	0	1
0	0,5	1	0	1
0	1	1	0	1
0,5	0	0	0	1
0,5	0,5	1	0,5	1
0,5	1	1	0,5	1
1	0	0	0	1
1	0,5	0,5	0,5	1
1	1	1	1	1

3.11. Uma vez que a coluna à direita tem somente uns, o *"modus ponens"* é válido para lógica fuzzy. A validade é limitada ao domínio escolhido, $(0,\ 0,5,\ 1)$, contudo esta validade pode ser estendida e o teste realizado no caso de maior dimensão.

Exemplo (basquete fuzzy): Vamos agora modificar o exemplo de basquete para examinar a diferença introduzida pela lógica fuzzy. A sentença dada pela expressão (3.3) contém cinco variáveis, mas em lógica fuzzy cada variável pode assumir, digamos, três valores. Isto significa que há $3^5 = 243$ combinações a considerar; 148 dessas combinações são legais no sentido que a sentença é verdadeira para estas combinações. Há outros casos onde há uma possibilidade de $0,5$ de vencer a aposta dependendo das possibilidades de vitória e derrota dos *"corredores"*, etc. Se estivermos novamente interessados nas combinações para as quais existe alguma possibilidade de se vencer a aposta, ou seja, $b \in \{0,\ 0,5\}$, então restam 88 combinações possíveis. Em vez de listarmos todas elas, vamos mostrar apenas uma para ilustração:

$$(p, c, g, d, b) = [0 \quad 0,5 \quad 0,5 \quad 1 \quad 0,5]$$

O exemplo mostra que lógica fuzzy traz outras soluções e requer mais esforço computacional do que no caso de lógica de dois valores. Este é o preço que pagamos para termos valores verdade intermediários que capturam a incerteza.

Implicação

A regra *"se o nível é baixo, então abra V_1"* é chamada de *"implicação"*, pois o valor de *"nível"* implica o valor de V_1 no controlador. É incomum, entretanto, usar a implicação de Gödel (3.4) em controladores fuzzy. Outra implicação que é usada com freqüência é a implicação de *Mamdani* [9].

Definição 5 *(Implicação de Mamdani) Sejam a e b dois conjuntos fuzzy, não necessariamente com o mesmo universo. A implicação de Mamdani é definida por:*

$$a \Rightarrow b \equiv a \bullet\cdot \min b \tag{3.5}$$

onde $\bullet\cdot \min$ é o "produto externo", correspondendo à aplicação de min *a cada elemento do produto Cartesiano entre a e b.*

Tabela 3.12: Produto externo

$\bullet \cdot \min$	b_1	b_2	$\ldots$	b_m
a_1	$a_1 \wedge b_1$	$a_1 \wedge b_2$	$\ldots$	$a_1 \wedge b_m$
a_2	$a_2 \wedge b_1$	$a_2 \wedge b_2$	$\ldots$	$a_2 \wedge b_m$
$\vdots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
a_n	$a_n \wedge b_1$	$a_n \wedge b_2$	$\ldots$	$a_n \wedge b_m$

Tabela 3.13: Exemplo de produto externo: *"baixo"* $\bullet \cdot \min$ *"abrir"*

$\bullet \cdot \min$	0	0,5	1
1	0	0,5	1
0,75	0	0,5	0,75
0,5	0	0,5	0,5
0,25	0	0,25	0,25
0	0	0	0

Seja a representado por um vetor coluna e b representado por um vetor linha, então o produto externo com mínimo pode ser obtido através de uma *"multiplicação de tabela"*. Esta operação está ilustrada na Tab. 3.12.

Exemplo (produto externo): Considere a implicação *"se o nível é baixo, então abra* V_1*"*, com *"baixo"* e *"abrir"* definidos como:

$$baixo = [1,\ 0,75,\ 0,5,\ 0,25,\ 0]$$
$$abrir = [0,\ 0,5,\ 1]$$

O produto externo está ilustrado na Tab. 3.13.

O produto externo baseado no operador **min** [9], bem como o produto externo com **min** substituído por * para multiplicação, é a base da maioria dos controladores fuzzy.

Inferência

Para se chegar a conclusões a partir de uma base de regras, é necessário um mecanismo que produza uma saída a partir de uma coleção de regras do tipo *"se-então"*. Isto é conhecido como *"inferência composicional de regras"*. O verbo *"inferir"* significa concluir a partir de evidências, deduzir ou ter uma conseqüência lógica. Para entender este conceito, é útil pensarmos em uma função $y = f(x)$, onde f é uma dada função, x é a variável independente e y é o resultado. O valor y_0 é inferido a partir de x_0 com a função f.

Uma regra famosa é o *"modus ponens"*:

$$(a \wedge (a \Rightarrow b)) \Rightarrow b \tag{3.6}$$

Se a sentença $a \Rightarrow b$ é verdadeira e a também é verdadeiro, então podemos inferir que b é verdadeiro. Lógica fuzzy generaliza este princípio com o *"modus ponens generalizado"*:

$$(a' \wedge (a \Rightarrow b)) \Rightarrow b' \tag{3.7}$$

Note que na lógica fuzzy a' e b' podem ser ligeiramente diferentes de a e b, por exemplo, por meio da aplicação de modificadores. O *"modus ponens generalizado"* está relacionado com encadeamento, isto é, raciocínio na direção direta das regras contidas na base de regras. Isto é particularmente útil em controladores fuzzy. O *"modus ponens generalizado"* é baseado na regra composicional de inferência.

Exemplo (*"modus ponens generalizado"*): dada a relação $R = \mathit{baixo} \bullet \cdot \min \mathit{abrir}$ do exemplo anterior e um vetor de entrada:

$$\mathit{nível} = [0,75,\ 1,\ 0,75,\ 0,5,\ 0,25] \tag{3.8}$$

então $V_1 = \mathit{nível} \vee \cdot \wedge R$, com R dada pela Tab. 3.13, produz o vetor:

$$[0\ 0,5\ 0,75] \tag{3.9}$$

A entrada *"nível"* dada em (3.8) é um conjunto fuzzy que representa o nível um pouco acima de *"baixo"*. O resultado após realizar inferência é um vetor V_1 ligeiramente abaixo de *"aberto"* conforme mostra (3.9). Se tentássemos colocar *"nível=baixo"*, esperaríamos obter um vetor V_1 com valor *"aberto"* após realizar a composição com R.

Múltiplas regras

Uma base de regras contém várias regras. *Como podemos combiná-las?* Considere a base de regras simples dada por:

$$\begin{cases} \text{Se o nível é baixo então abra } V_1 \\ \text{Se o nível é alto então feche } V_1 \end{cases} \tag{3.10}$$

Implicitamente assumimos o operador lógico *"ou"* entre as regras, tal que a base de conhecimento consiste de $R_1 \vee R_2$, com R_1 representando a primeira regra e R_2, a segunda. As regras são equivalentes às matrizes de implicação $\mathbf{R}_1$ e $\mathbf{R}_2$, portanto, a regra total é obtida através da operação lógica *"ou"* de duas tabelas, item por item, conforme:

$$\mathbf{R} = \vee R_i$$

Inferência pode então ser aplicada sobre $\mathbf{R}$.

3.5 Introdução a sistemas de controle fuzzy

Embora a lógica fuzzy tenha se tornado bem conhecida na área de controle automático, há reações adversas que consideraram a lógica fuzzy como uma lógica pouco formalizada. Talvez outra definição desta lógica teria sido mais apropriada.

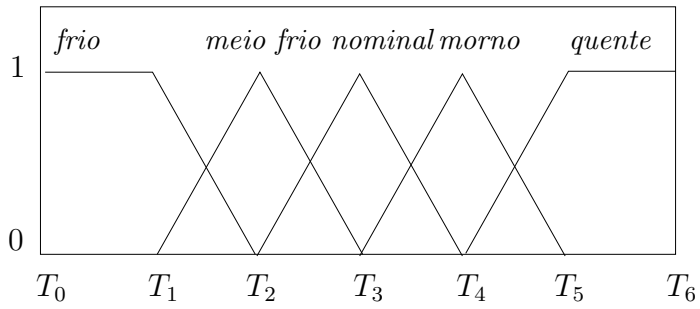

Figura 3.6: Funções de pertinência para conjuntos fuzzy de temperatura.

A lógica fuzzy pode ser vista como uma forma de se realizar uma interface com processos analógicos, que permite o tratamento gradual de sinais contínuos até sinais discretos manipulados por máquinas digitais.

Como exemplo, considere um sistema de freios ABS dirigido por um micro-controlador. O micro-controlador deve tomar decisões baseado na temperatura dos freios, velocidade e outras variáveis do sistema. A variável *"temperatura"* pode ser categorizada em faixas de estados, tais como: *"frio"*, *"meio frio"*, *"nominal"*, *"morno"* e *"quente"*. A definição da fronteira entre estes estados não é clara. Um limiar poderia ser definido para diferenciar *"morno"* de *"quente"* mas isto levaria a uma mudança descontínua quando o valor de entrada atravessa o limiar. A alternativa é assumir estados fuzzy, isto é, fazer com que os estados mudem gradualmente de um estado para o próximo. Poderíamos definir estados de temperatura usando *"funções de pertinência"* tais como as indicadas na Fig. 3.6.

Com este esquema, os estados das variáveis de entrada não tem saltos abruptos de um estado para o próximo. Ao contrário, à medida que a temperatura muda, ela perde valor de pertinência em um conjunto fuzzy enquanto aumenta o grau de pertinência ao próximo conjunto fuzzy. Em qualquer instante, o valor da temperatura dos freios será um grau de pertinência com respeito a dois conjuntos fuzzy: 60% *"nominal"* e 40% *"morno"* ou 70% *"nominal"* e 30% *"meio frio"*, por exemplo.

As variáveis de entrada em um sistema de controle fuzzy são em geral mapeadas para o sistema por meio de funções de pertinência semelhantes às vistas acima. O processo de conversão de valores de entrada precisos para valores fuzzy é conhecido por *"fuzificação"*.

Um sistema de controle pode também ter vários tipos de chaves (*"switches"*), ou entradas *"on-off"*, além de entradas analógicas, e tais chaves sempre terão como valor 1 ou 0. O esquema proposto acima pode tratá-las como funções fuzzy simplificadas que assumem valor 0 ou 1.

Dados *"mapeamentos"* das variáveis de entrada para funções de pertinência e valores verdade, um micro-controlador então pode tomar decisões sobre a ação a ser tomada baseado em um conjunto de *"regras"*, cada uma delas da forma:

> **Se** *temperatura (do freio)* **é** *morna* **e** *velocidade* **é** *não muito rápida*
> **então** *pressão (do freio)* **é** *ligeiramente reduzida*

Neste exemplo, as duas variáveis de entrada são *"temperatura (do freio)"* e *"velocidade"* cujos valores são definidos como conjuntos fuzzy. A variável de saída,

"pressão (do freio)", é também definida como um conjunto fuzzy que pode assumir valores como *"estática"*, *"ligeiramente aumentada"*, *"ligeiramente reduzida"* e assim por diante.

A regra acima parece estranha já que aparentemente a mesma poderia ser utilizada sem fazer uso da lógica fuzzy, mas devemos lembrar que a decisão é baseada em um conjunto de regras:

- Todas as regras aplicáveis são invocadas utilizando funções de pertinência e valores verdade obtidos a partir das entradas, com o objetivo de se chegar ao resultado da regra.
- Este resultado será mapeado em uma função de pertinência e valor verdade para controle da variável de saída.
- Estes resultados são combinados para dar uma resposta exata (do inglês, *"crisp"*), ou seja, o valor exato da pressão, em um processo conhecido como *"defuzificação"*.

A combinação de operações fuzzy e inferência baseada em regras define um *"sistema especialista fuzzy"*.

Sistemas de controle clássico são baseados em um modelo matemático segundo o qual o sistema de controle é descrito através de uma ou mais equações diferenciais, as quais definem a resposta do sistema às entradas. Tais sistemas são freqüentemente implementados como controladores PID (*proportional-integral-derivative*). Eles são produto de décadas de análise teórica e desenvolvimento tecnológico, sendo altamente eficazes.

Se controladores PID e outras técnicas de controle clássico são tão bem desenvolvidos, por quê devemos nos preocupar com controle fuzzy? Controle fuzzy tem algumas vantagens. Em vários casos, o modelo matemático do processo não existe, não é conhecido ou é muito complexo de ser obtido ou implementado em máquinas computacionais. Em tais situações, um sistema baseado em regras empíricas pode ser mais eficaz.

Além disso, lógica fuzzy é bem estudada e apropriada para implementações de baixo custo computacional, com sensores de baixo custo e utilizando conversores analógico-digitais de baixa resolução. Tais sistemas também podem ser expandidos/aperfeiçoados ao se adicionar regras que venham a melhorar o desempenho. Em muitos casos, controle fuzzy pode ser usado para melhorar sistemas de controle já existentes, adicionando uma camada de controle inteligente acima do método de controle atual.

3.6 Controle fuzzy em detalhes

O projeto conceitual de controladores fuzzy é bastante simples. Tais controladores consistem de um estágio de entrada (*"fuzificação"*), um estágio de processamento (*"inferência"*) e um estágio de saída (*"defuzificação"*), conforme ilustra a Fig. 3.7. O estágio de entrada mapeia sensores e outras entradas (como chaves) para funções de pertinência apropriadas e valores verdade. O estágio de processamento invoca as

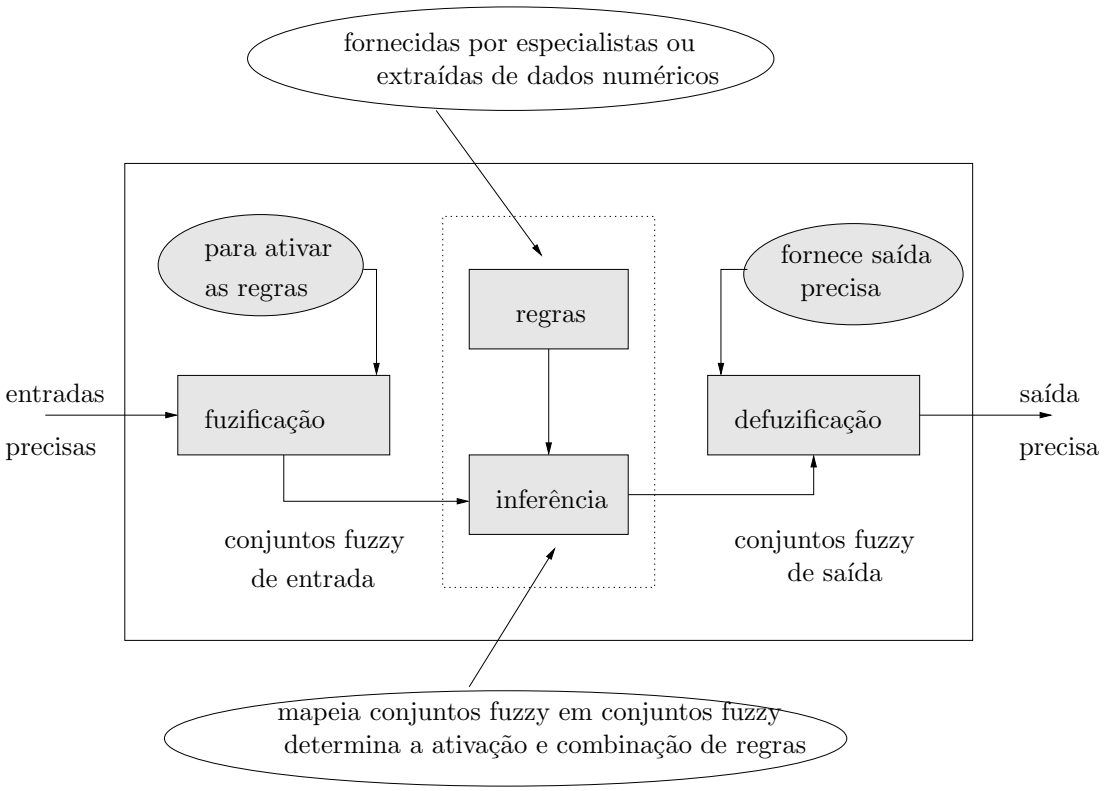

Figura 3.7: Arquitetura de um controlador fuzzy [16].

regras apropriadas e gera os resultados para cada uma, depois combina os resultados das regras. Finalmente, o estágio de saída converte o resultado da combinação em um valor de saída preciso para controle.

O formato de função de pertinência mais comum é o triangular, embora curvas trapezoidais e tipo sino sejam também empregadas, o número e localização das curvas desempenha papel mais crítico. Muitas vezes, um número entre três e sete curvas é suficiente para cobrir a faixa dos valores de entrada, ou o *"universo de discurso"* dentro do jargão da lógica fuzzy.

O estágio de processamento é baseado em uma coleção de regras lógicas da forma **se-então**, onde a parte do **se** é chamada de *"antecedente"* e a parte do **então** é conhecida por *"conseqüente"*. Sistemas de controle fuzzy tipicamente têm dezenas de regras. Considere a regra para um termostato:

se (temperatura é *"fria"*) **então** (aquecedor é *"alto"*)

Esta regra usa o valor verdade da *"temperatura"* de entrada, que é um certo valor de *"fria"*, para gerar um resultado em um conjunto fuzzy para a saída do *"aquecedor"*, que é um valor *"alto"*. Este resultado é utilizado com o resultado de outras regras para gerar um valor composto exato para a saída (do inglês, *"crisp composite output"*). Obviamente, quanto maior o grau de verdade de *"fria"*, maior o grau de verdade de *"alto"*, embora isto não significa necessariamente que a saída fixará o aquecedor no estado *"alto"*, já que esta é apenas uma entre várias regras.

Em alguns casos, as funções de pertinência podem ser alteradas por meio de *"modificadores"* que são equivalentes a adjetivos. Modificadores típicos incluem

"em torno", *"próximo de"*, *"muito"*, *"ligeiramente"*, *"extremamente"* e *"um pouco"*, entre outros. Essas operações podem ter definições precisas, como visto na seção 3.3, mas podem variar de uma implementação para outra. Como visto acima, o modificador *"muito"* eleva ao quadrado o valor da função de pertinência — uma vez que as funções de pertinência tem valor sempre menor que 1, o efeito é de estreitar a função de pertinência. O modificador *"extremamente"* eleva ao cubo a função de pertinência, estreitando ainda mais a função de pertinência. Por outro lado, o modificador *"um pouco"* alarga a função de pertinência tomando a raiz quadrada.

Na prática, as regras fuzzy têm vários antecedentes que são combinados usando operadores fuzzy, tais como *"e"*, *"ou"* e *"não"*, mesmo que as definições possam variar:

e: em uma definição bem popular, simplesmente assume o menor valor verdade de seus antecessores; e

não: este operador tipicamente subtrai o valor da função de pertinência de 1, dando dessa forma a função *"complemento"*.

Há diferentes maneiras de se definir o resultado de uma regra, mas uma das mais comuns e mais simples é por meio do método de inferência *"max-min"*, a qual passa os valores verdade gerados pelas premissas à função de pertinência de saída.

Regras podem ser resolvidas em paralelo via hardware ou seqüencialmente via software. Os resultados de todas essas regras que foram disparadas são *"defuzificados"* para valores exatos por meio de um dentre vários métodos. Há dezenas de métodos propostos na teoria, cada um com vantagens e desvantagens.

O método *"centróide"* é bastante popular, no qual o *"centro de massa"* do resultado provê o valor exato. Outro método é o método da *"altura"*, que toma o valor da maior contribuição. O método *"centróide"* favorece a regra com maior área, enquanto o método da *"altura"* obviamente dá preferência à saída de maior valor.

O exemplo a seguir ilustra a inferência *"max-min"* e defuzificação por meio do método *"centróide"* para um sistema com variáveis de entrada x, y e z, além de uma variável de saída ψ. Os procedimentos de inferência e de defuzificação são ilustrados na Fig. 3.8. Note que μ é padrão em lógica fuzzy para designar valor verdade. Observe ainda que cada regra dá um valor verdade para uma função de pertinência de uma variável de saída. Na defuzificação *"centróide"*, os valores são combinadores via operador *"ou"*, ou seja, o valor máximo é usado e estes não são adicionados, os resultados são então processados por meio do cálculo centróide.

Sistemas de controle fuzzy são baseados em métodos empíricos, basicamente um método metódico de tentativa e erro. O processo geral pode ser caracterizado pelos passos a seguir:

- documente as especificações operacionais do sistema, as entradas e saídas.
- determine conjuntos fuzzy para as entradas.
- defina um conjunto de regras.
- escolha um método de defuzificação.

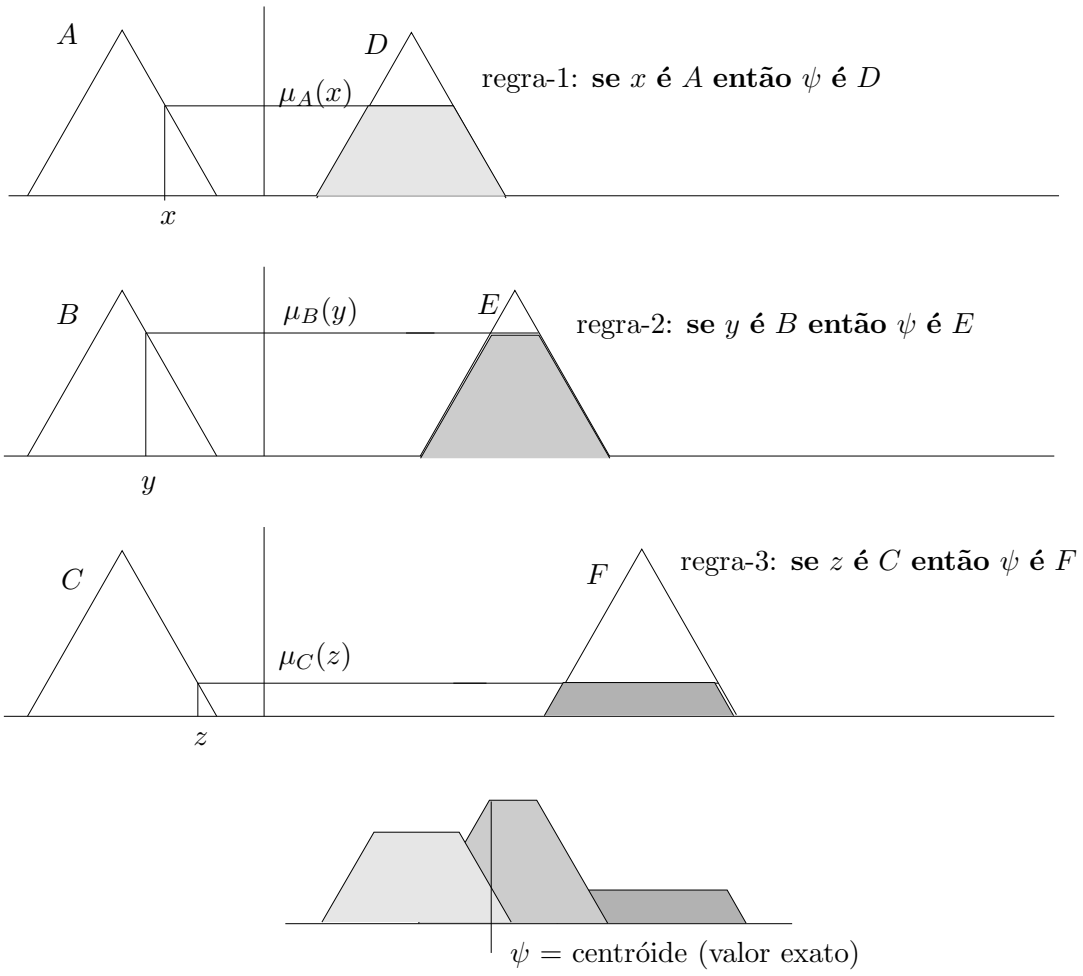

Figura 3.8: Inferência *max-min*.

- execute o sistema sobre um conjunto de teste, fazendo ajustes quando necessário.
- finalize a documentação e implemente o sistema de controle.

Como um exemplo, considere o projeto de um controlador fuzzy para uma turbina. O diagrama de blocos do sistema de controle pode ter a forma dada na Fig. 3.9. Há duas variáveis de entrada (temperatura e pressão) e uma variável de saída (a abertura da válvula). A operação da turbina pode ser revertida, logo a abertura da válvula pode ser positiva ou negativa. Os mapeamentos em conjuntos fuzzy estão indicados na Fig. 3.10.

A abertura da válvula é regulada pelos valores:

- N_3: negativo alto.
- N_2: negativo médio.
- N_1: negativo pequeno.
- Z: zero.

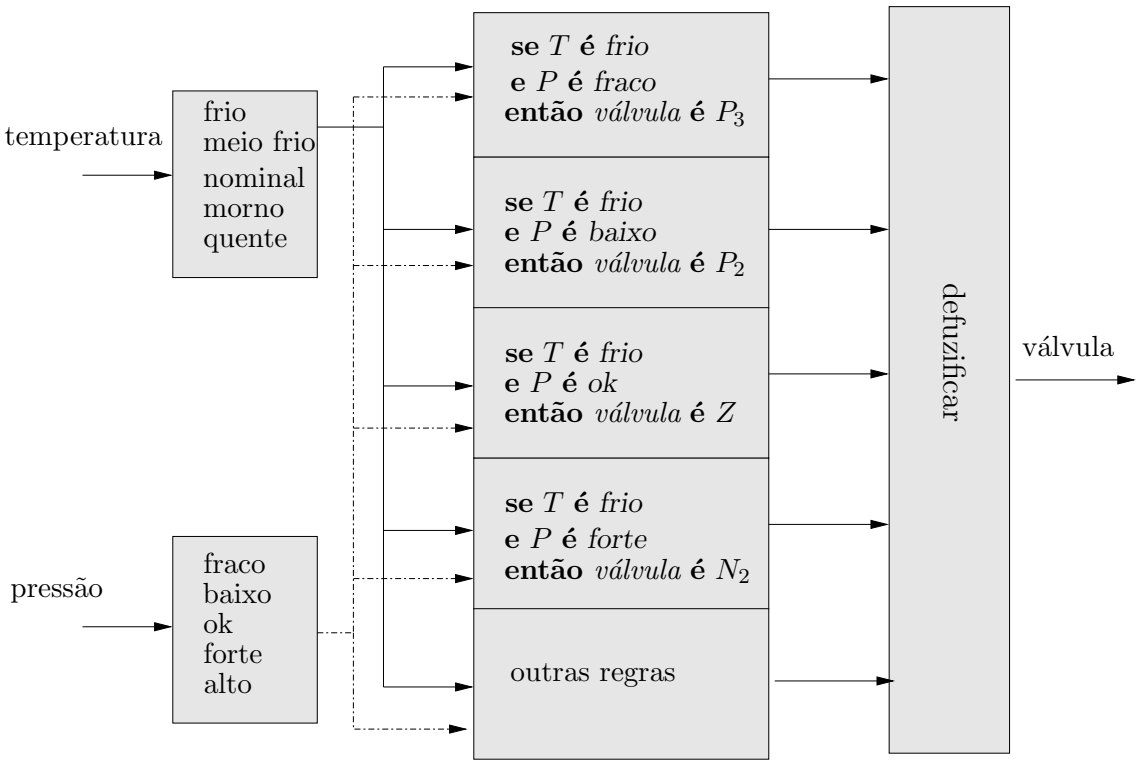

Figura 3.9: Diagrama de blocos do controlador fuzzy para turbina.

- P_1: positivo pequeno.
- P_2: positivo médio.
- P_3: positivo grande.

O conjunto de regras inclui regras tais como:

- Regra 1: **se** *temperatura* **é** *fria* **e** *pressão* **é** *fraca* **então** *válvula* **é** P_3.
- Regra 2: **se** *temperatura* **é** *fria* **e** *pressão* **é** *baixa* **então** *válvula* **é** P_2.
- Regra 3: **se** *temperatura* **é** *fria* **e** *pressão* **é** *ok* **então** *válvula* **é** Z.
- Regra 2: **se** *temperatura* **é** *fria* **e** *pressão* **é** *forte* **então** *válvula* **é** N_2.

Na prática, o controlador aceita entradas e os mapeia em valores verdade por meio das funções de pertinência. Então esses mapeamentos alimentam as regras. Se a regra estabelece uma operação *"e"* entre os mapeamentos de duas variáveis de entrada, como os exemplos acima ilustram, o menor valor das duas é utilizado como valor verdade da combinação. Se a regra estabelece uma operação *"ou"*, então o máximo é utilizado. A saída apropriada é escolhida e designada um valor de pertinência no nível de verdade da premissa. Os valores verdade são então defuzificados.

Como exemplo, suponha que a temperatura está no estado *"fria"*, e que a pressão está nos estados *"baixo"* e *"ok"*. Os valores de pressão garantem que apenas as regras 2 e 3 irão disparar. A regra 2 é avaliada conforme ilustra a Fig. 3.11. A

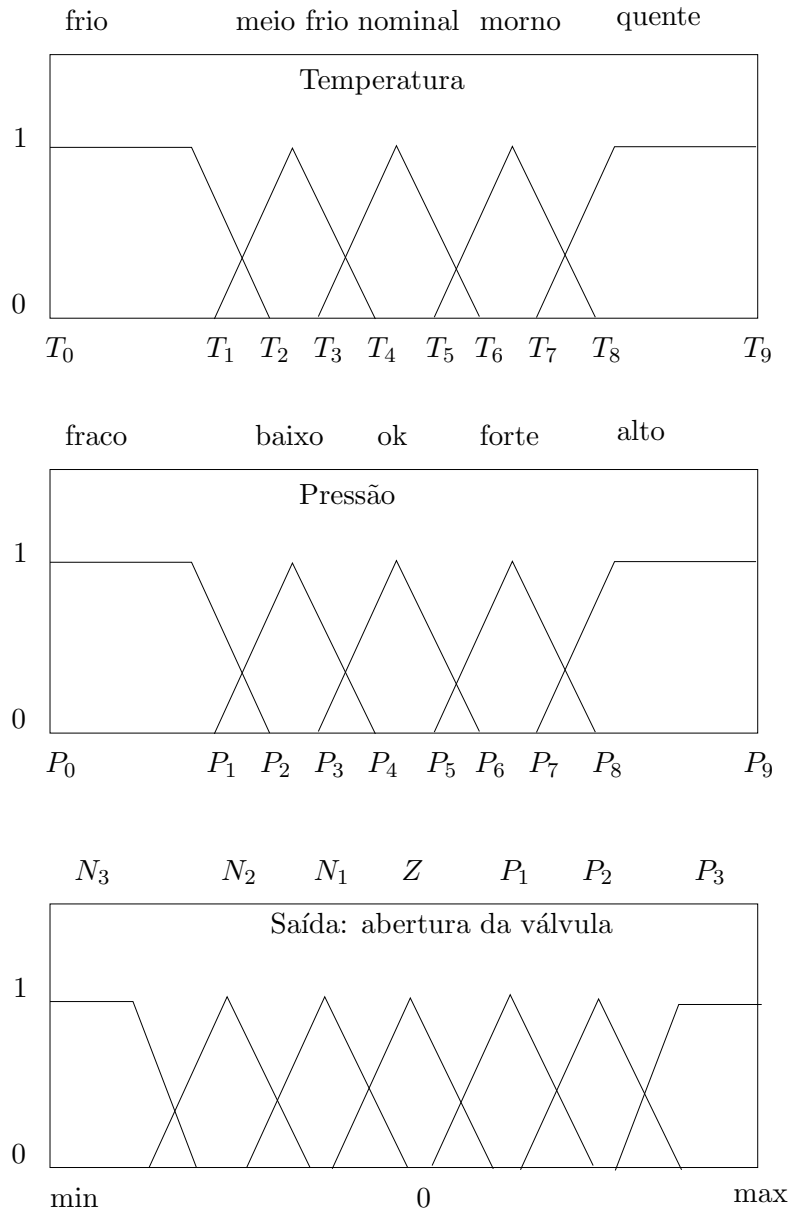

Figura 3.10: Procedimento de fuzificação para o controle fuzzy da turbina.

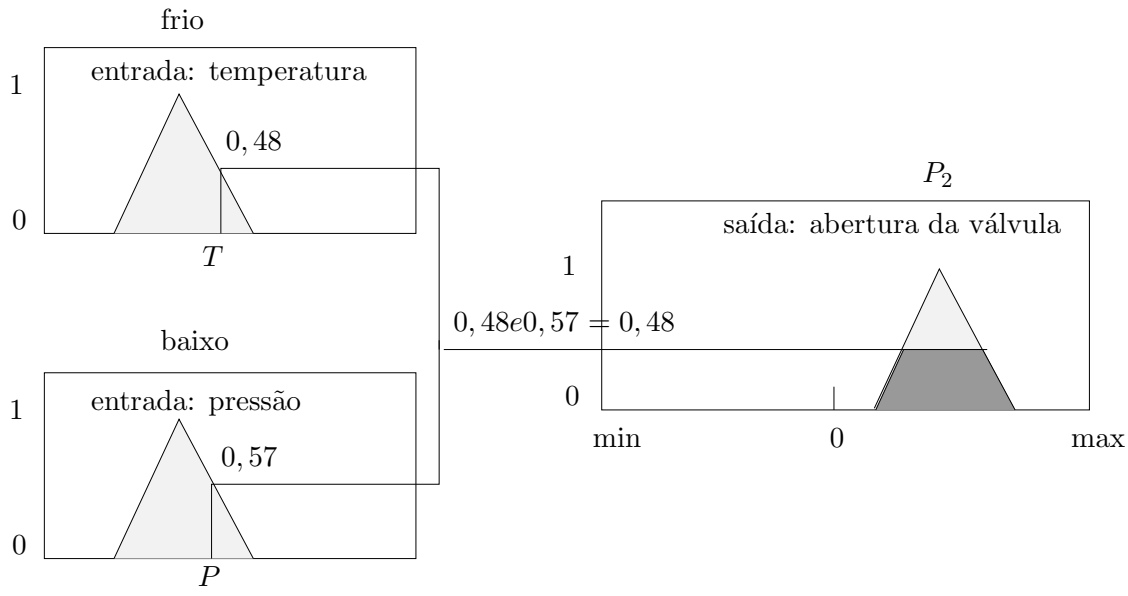

Figura 3.11: Disparo da regra 2.

regra 3 é avaliada conforme ilustra a Fig. 3.12. As saídas das regras 2 e 3 são então combinadas como mostra a Fig. 3.13. A saída combinada será então implementada, ajustando a abertura da válvula, e o ciclo de controle será então repetido para produzir o próximo valor.

3.7 Implementação de um controlador fuzzy

Considere a implementação de um controlador realimentado (*feedback controller*) simples. Um conjunto fuzzy é definido para a entrada (variável de erro) *"erro"*, para a variação derivada do erro a partir do erro anterior (variável *"delta"*) bem como a variável de saída (*"saída"*), como segue:

- LP: valor positivo alto
- SP: valor positivo pequeno
- ZE: zero
- SN: valor negativo pequeno
- LN: valor negativo grande

Se o erro varia de -1 a $+1$, com o conversor analógico digital tendo precisão de $0,25$, então o conjunto fuzzy para as variáveis de entrada (e, neste caso, também para a variável de saída) pode ser descrito através de uma tabela, onde os valores de *"erro"*, *"delta"* e *"saída"* aparecem na linha superior, enquanto os valores das funções de pertinência são dados nas linhas abaixo, como mostra a Tab. 3.14.

A Tab. 3.14 simplesmente dá o valor de pertinência para cada conjunto fuzzy (*"LP"* a *"LN"*), em termos dos valores μ com relação aos valores dados na linha superior.

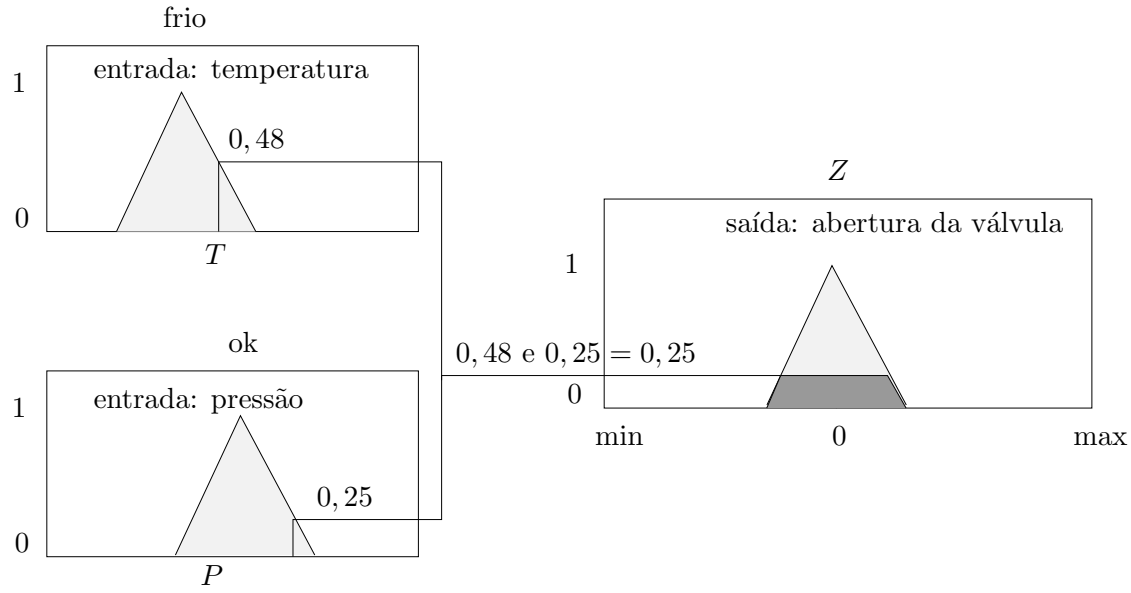

Figura 3.12: Disparo da regra 3.

Tabela 3.14: Funções de pertinência

$\mu \backslash$ *variável*	-1	-0,75	-0,50	-0,25	0	0,25	0,50	0,75	1
μ_{LP}	0,0	0,0	0,0	0,0	0,0	0,0	0,3	0,7	1,0
μ_{SP}	0,0	0,0	0,0	0,0	0,3	0,7	1,0	0,7	0,0
μ_{ZE}	0,0	0,0	0,3	0,7	1,0	0,7	0,3	0,0	0,0
μ_{SN}	0,3	0,7	1	0,7	0,3	0,0	0,0	0,0	0,0
μ_{LN}	1,0	0,7	0,3	0,0	0,0	0,0	0,0	0,0	0,0

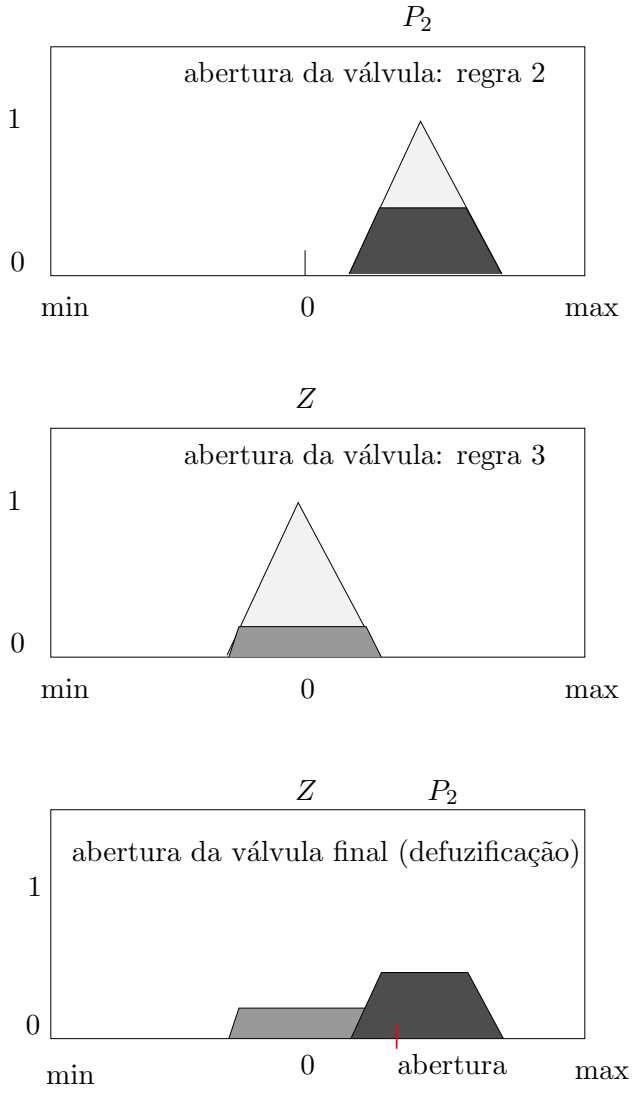

Figura 3.13: Defuzificação.

É importante salientar que para as variáveis *"erro"* e *"delta"* os valores da linha superior são usados para gerar os valores μ, enquanto que para a variável de saída, os valores μ são usados para gerar os valores da linha superior.

Suponha que o sistema fuzzy tenha a seguinte base de regras fuzzy:

- regra 1: **se** *erro* = *ZE* **e** *delta* = *ZE* **então** *saída* = *ZE*
- regra 2: **se** *erro* = *ZE* **e** *delta* = *SP* **então** *saída* = *SN*
- regra 3: **se** *erro* = *SN* **e** *delta* = *SN* **então** *saída* = *LP*
- regra 4: **se** *erro* = *LP* **ou** *delta* = *LP* **então** *saída* = *LN*

Estas regras são típicas de aplicações de controle, onde os antecedentes consistem da combinação lógica dos sinais de *"erro"* de *"delta"*, enquanto o conseqüente é um comando de controle.

As regras podem ser um pouco difíceis de serem interpretadas. Por exemplo, o que a regra 1 diz é que se o valor de *"erro"* pertence ao conjunto fuzzy *ZE* e o sinal *"delta"*, ou variação do último sinal de erro, pertencer ao conjunto fuzzy *ZE*, então a saída de controle também pertence ao conjunto fuzzy *ZE*. O que torna confuso é que a atribuição de valores numéricos a cada regra não dá, pelo menos diretamente, um valor preciso para o sinal de saída.

O valor de saída é definido pela regra 1 como *ZE*. Analisando os valores fuzzy, a tabela mostra que *ZE* tem seu máximo para o valor "0" da linha superior da tabela e, portanto, a regra sempre dá uma saída 0. Similarmente, a regra 2 dá uma saída para *SN* ou $-0,5$, a regra 3 sempre dá saída *LP* ou 1 e a regra 4 dá a saída *LN* ou -1.

Devemos nos lembrar que o valor de saída final é uma *"combinação"* das saídas dessas quatro regras, ponderadas por meio de valores μ apropriados e computados de acordo com as regras de cálculo usando o método centróide:

$$\frac{\sum_{i=1}^{4} \mu(i) saída(i)}{\sum_{i=1}^{4} \mu(i)}$$

onde $\mu(i)$ é o valor verdade da regra i, cujo cômputo é ilustrado nas Figs. 3.11 e 3.12. A expressão acima consiste do processo de defuzificação exemplificado na Fig. 3.13.

Suponha que em um certo momento tenhamos:

$$erro = 0,25$$
$$delta = 0,5$$

Então os valores μ podem ser obtidos diretamente da Tab. 3.14, coletando os valores das colunas apropriadas, o que produz os valores da Tab. 3.15.

Estes valores são repassados às regras para se obter os valores de saída. Para a regra 1, temos:

Tabela 3.15: Funções de pertinência para o caso $erro = 0,25$ $delta = 0,5$

μ\variável	*erro=0,25*	*delta=0,50*
μ_{LP}	0,0	0,3
μ_{SP}	0,7	1,0
μ_{ZE}	0,7	0,3
μ_{SN}	0,0	0,0
μ_{LN}	0,0	0,0

regra 1: **se** *erro=ZE* **e** *delta=ZE* **então** *saída=ZE*

A ponderação para a saída, "$\mu(1)$", é produzida com o simples cálculo:

$$\mu(1) = \min(0,7,\ 0,3) = 0,3$$

E como mencionado acima, a saída desta regra é sempre 0:

$$\text{saída}(1) = 0$$

As outras regras nos dão:

regra 2: **se** *erro=ZE* **e** *delta=SP* **então** *saída=SN*
$\mu(2) = \min(0,7,\ 1) = 0,7$
saída(2) = -0,5
regra 3: **se** *erro=SN* **e** *delta=SN* **então** *saída=LP*
$\mu(3) = \min(0,0,\ 0,0) = 0,0$
saída$(3) = 1$
regra 4: **se** *erro=LP* **ou** *delta=LP* **então** *saída=LN*
$\mu(4) = \max(0,0,\ 0,3) = 0,3$
saída(4) = -1

A computação do centróide então leva aos valores:

$$\begin{aligned}
\text{saída} &= \frac{\mu(1)\text{saída}(1) + \mu(2)\text{saída}(2) + \mu(3)\text{saída}(3) + \mu(4)\text{saída}(4)}{\mu(1) + \mu(2) + \mu(3) + \mu(4)} \\
&= \frac{0,3 \times 0 + 0,7 \times -0,5 + 0 \times 1 + 0,3 \times -1}{0,3 + 0,7 + 0 + 0,3} \\
&= \frac{0 - 0,35 + 0,0 - 0,3}{1,3} \\
&= \frac{-0,65}{1,3} \\
&= -0,5
\end{aligned}$$

Contudo, a parte mais difícil está em descobrir quais regras produzem bons resultados na prática.

Para entender o cômputo do centróide, lembramos que o centróide é definido como a soma de todos os momentos (localização vezes massa) em torno do centro de

gravidade e forçando a soma para zero (a soma dos momentos deve ser nula). Então se x_0 é o centro de gravidade, x_i a localização de cada massa e m_i a massa, isto nos dá a equação:

$$\begin{aligned} 0 &= (x_1 - x_0) * m_1 + (x_2 - x_0) * m_2 + \ldots, (x_n - x_0) * m_n \\ &= (x_1 m_1 + x_2 m_2 + \ldots + x_n m_n) - x_0(m_1 + m_2 + \ldots + m_n) \end{aligned}$$

e daí segue que:

$$x_0(m_1 + m_2 + \ldots + m_n) = x_1 m_1 + x_2 m_2 + \ldots + x_n m_n$$

assim chegamos à equação:

$$x_0 = \frac{x_1 m_1 + x_2 m_2 + \ldots + x_n m_n}{m_1 + m_2 + \ldots + m_n}$$

No nosso caso, os valores μ correspondem às massas e os valores de x correspondem às saídas das regras.

3.8 Referências

O material apresentado neste capítulo é uma compilação traduzida de relatórios técnicos escritos por Jan Jantzen [8] e Greg Goebel [5]. Outras referências introdutórias para lógica fuzzy incluem os artigos de Kevin Self [15] e de Earl Cox [2].

Capítulo 4

Computação Evolutiva

"How can computers be programmed so that problem-solving capabilities are built up by specifying 'what is to be done' rather than 'how to do it?'." [John Holland, 1975]

4.1 Introdução à computação evolutiva

John Holland [7] indagou sobre algumas questões que levaram à concepção dos algoritmos genéticos durante os anos 60:

- *"Como que a evolução produz organismos cada vez mais hábeis em ambientes que são altamente incertos para um organismo particular?"*
- *"Como que um organismo faz uso da experiência para modificar seu comportamento de maneira que lhe traga benefícios?"*
- *"Como que computadores podem ser programados com a capacidade de resolver problemas, dizendo o que deve ser feito em vez de como fazê-lo?"*

Em essência, foi demonstrado que os problemas intrínsecos a estas questões podem ser reduzidos a um problema de otimização de funções multivariadas. O *"o problema da Natureza"* está em criar organismos capazes de se reproduzir (mais aptos) em um ambiente particular: o ambiente determina as regras de seleção e as soluções deste processo de solução são os organismos. Na linguagem de otimização, as soluções para um problema particular (um problema de engenharia, por exemplo) serão selecionadas com base em quão bem elas resolvem o problema. Algoritmos genéticos são inspirados no processo de seleção das soluções para o problema, não sendo computadas algebricamente. A solução é encontrada através de uma população de soluções candidatas que são alteradas pelo algoritmo a cada passo com o intuito de aumentar a probabilidade de se gerar melhores soluções na população. Em outras palavras, algoritmos genéticos e outros métodos evolucionários (ME), tal como programação evolutiva (PE), exploram um espaço multidimensional de soluções alternativas para um problema particular. Esta exploração se dá através de uma população de cadeias que codificam as soluções, as quais são submetidas a processos de variação (*"crossover"* e *"mutação"*) e são reproduzidas de forma a levar a população para regiões mais promissoras do espaço de busca (*"seleção"*).

4.1.1 Estratégias evolucionárias e auto-organização

O princípio fundamental dos métodos evolucionários está na separação de soluções para um problema particular (*máquina*) da descrição destas soluções (*memória*). Algoritmos genéticos operam nestas descrições e não nas soluções propriamente ditas, isto é, variações são aplicadas nas descrições, enquanto as respectivas soluções são avaliadas e as descrições de soluções são selecionadas com base nestas avaliações. Este aspecto de separação entre máquina e descrição segue o esquema de auto-reprodução de von Neumann que permitiu elevar a complexidade das máquinas.

Contudo, a forma de organização praticada por algoritmos genéticos não se auto-organiza como a rede Booleana de um autômato celular. Apesar das soluções serem obtidas por meio da interação dos elementos de uma população, e seguindo as regras gerais usualmente observadas por sistemas computacionais emergentes, os algoritmos genéticos não se auto-organizam, pois são baseados em processos seletivos pressionados por funções de aptidão (*"fitness functions"*). A ordem atingida não é um resultado da dinâmica interna de uma coleção de elementos que interagem (tal como em uma rede aleatória), mas o resultado de um critério de seleção externo. Neste sentido dizemos que os métodos evolutivos seguem um esquema de organização seletiva baseado em memória.

Por outro lado, auto-organização é atribuída aos comportamentos de organizações que são totalmente dependentes da dinâmica interna. Isto é usualmente visto em termos de atratores para comportamentos de sistemas dinâmicos determinados por estados. Métodos evolutivos tem fundamentação nos seguintes conceitos:

- a dicotomia entre solução e descrição que permite introduzir o conceito de memória;
- as transições em sistemas evolutivos não são determinadas por estados, sendo decorrentes de variações estocásticas; e
- o processo de seleção é externo à população de elementos (descrições de soluções).

Dessa forma, não se pode dizer que uma população de memórias está interagindo com base em alguma *"auto-dinâmica:"* as solução alcançadas por um algoritmo genético não se auto-organizam mas são produzidas via variação estocástica da população e mecanismos de solução externos.

Sistemas capazes de desenvolver mecanismos para tirar vantagem desta variação baseada na dicotomia solução-descrição podem seguir um tipo de seleção baseada no princípio de auto-organização semântica. Contudo, algoritmos genéticos computacionais não são baseados neste princípio, a relação de codificação é externamente definida e não evolui com o sistema. Por causa das razões acima é natural visualizar métodos evolucionários como totalmente distintos de sistemas auto-organizáveis. Talvez seja útil pensar que ME modelam um aspecto diferente de sistemas biológicos que está relacionado à seleção natural. Sistemas auto-organizáveis modelam características abstratas e internas, enquanto modelos de sistemas evolucionários modelam a existência de processos de seleção natural (externa) de memórias variantes baseados em uma descrição do sistema.

4.2 Algoritmo genético (AG)

4.2.1 Genética e evolução

A percepção de que certas características são hereditárias (i.e., transmitidas geneticamente) data de várias décadas passadas. A Teoria da Evolução [3] é considerada um dos maiores avanços científicos até esta data. Ela tem implicações em várias áreas naturais, científicas e sociais como, por exemplo, Biologia, Matemática, Física e Sociologia.

4.2.2 Adaptação biológica

Estudaremos maneiras pelas quais o processo de evolução natural pode ser empregado para se *"evoluir"* soluções computacionais (algoritmo genético) e até mesmo programas ou algoritmos (Programação Genética) para problemas interessantes.

A propriedade de adaptação é descrita pela seguinte expressão:

Adaptação = Variedade + Hereditariedade + Seleção

1) Variedade: Se refere a como os indivíduos diferem uns dos outros, portanto só pode ocorrer se existirem múltiplos indivíduos, implicando em paralelismo e multiplicidade espacial (*população*).

2) Hereditariedade: É uma forma de persistência temporal que se propaga de pai para filho, um processo iterativo que se repete (*processo iterativo*).

3) Seleção Natural: A capacidade de reprodução é exponencial, extremamente maior do que a quantidade de recursos disponíveis (limitada). Surge, portanto, a frase *"a sobrevivência dos reprodutores"*. Levando em conta esta capacidade exponencial, os seres humanos formam um grupo seleto de indivíduos (*Seleção*).

A frase mais conhecida é *"a sobrevivência do mais dotado"*. Entretanto, a teoria diz que o que importa é a reprodução, o que implica a frase *"a sobrevivência do reprodutor"*. Segundo a teoria da evolução, as características de uma pessoa (cor dos olhos, alta, inteligente, etc.) só importam se aumentam a capacidade de reprodução. A evolução das espécies pode levar a processos cíclicos como, por exemplo, a dinâmica entre leões e gazelas. Se os leões se tornam mais perspicazes ao caçarem gazelas, então as gazelas sobreviventes tendem a ser mais rápidas, que por sua vez vão gerar descendentes mais ágeis, dificultando a caça dos leões.

4.2.3 Hereditariedade com evolução simulada

Da mesma forma que em sistemas biológicos, as equações de adaptação biológica podem ser utilizadas para *"evoluir"* soluções de um problema (Algoritmo Genético) e até mesmo algoritmos (Programação Genética). John Holland [7] foi um dos primeiros a propor a simulação do processo de evolução natural para resolver problemas diversos, tais como problemas de otimização e de aprendizagem de máquina. Ele

concebeu o termo algoritmo genético (AG) e foi um dos grandes impulsionadores da área que hoje é conhecida como *computação evolucionária*.

O algoritmo genético representa soluções como estruturas semelhantes ao DNA, as quais expressam alguma forma de aptidão. Isto está inspirado na Natureza, onde estruturas de DNA aparecem em organismos com capacidade de se acasalarem, e onde haja mutação e *cross-over* do material genético. Estes princípios fundamentais foram apresentados através da teoria de Darwin, mais tarde complementada pelos trabalhos de Gregor Mendel (pai da Genética) e posteriormente enriquecidos pela descoberta da estrutura de dupla hélice do DNA. Estes últimos trabalhos mostram que as características transmitidas são discretas, não são contínuas como anteriormente assumido. Mendel mostrou em seus experimentos que, se controlarmos o ambiente, as características são bem distintas (e.g., flores altas ou baixas). Conclusão:

A linguagem da natureza é um alfabeto discreto.

O AG expressa uma solução como uma cadeia de símbolos (usualmente 0s e 1s) de tamanho fixo, da mesma forma que o DNA codifica as características de um indivíduo. É necessária a existência de uma função de aptidão (*fitness function*) que mapeie a cadeia em uma forma útil. Por exemplo, a cadeia pode representar:

- um vetor com as variáveis de uma função f que se deseja minimizar; ou
- uma estratégia para competir em um jogo.

4.2.4 Popularidade do algoritmo genético

A popularidade do algoritmo genético se dá em função de três propriedades:

1) Robustez: Evolução natural é tida como um método bem-sucedido e robusto no meio biológico.

2) Generalidade: AGs são capazes de tratar problemas complexos, que possuem um número grande de componentes e cujas contribuições para o todo não são bem entendidas.

3) Paralelização: AGs são naturalmente paralelos; podem ser implementados em redes de computadores (máquinas assíncronos) e computadores paralelos (máquinas síncronos).

Antes de procedermos à descrição dos passos do algoritmo genético, assumiremos o seguinte:

- O problema de otimização a ser resolvido é de maximização, em vez de minimização:

$$P: \quad \begin{array}{ll} \text{Minimize} & f(x) \\ x \in \mathbb{R}^n & \end{array}$$

onde nada se assume sobre $f : \mathbb{R}^n \to \mathbb{R}$

Para transformar minimização em maximização, basta substituir o operador *min* por *max* e $f(x)$ por $-f(x)$.

- A função objetivo ($f(x)$ se maximização, $-f(x)$ se minimização) será utilizada como função de aptidão (*fitness function*, f_i). Portanto, quanto mais alto o valor da *fitness function* (f_i) melhor a qualidade da respectiva solução. Assumiremos também que a *fitness function* assume um valor não negativo. Isso pode ser contornado através da adição de uma constante, i.e., seja $m = \min\{f_i(x) : x$ pertence à população$\}$, então basta substituir $f_i(x)$ por $f_i(x) - m$ para que a *fitness function* se torne não negativa.

4.2.5 Algoritmo genético em detalhes

O algoritmo genético pode ser descrito como uma função $AG(f_i, f_t, p, r, m)$ cujos parâmetros de entrada definem um problema a ser resolvido. Abaixo segue o pseudo-código.

Algoritmo $AG(f_i, f_t, p, r, m)$

Parâmetros de entrada:
f_i - *fitness function*
f_t - *fitness threshold* (condição de parada)
p - tamanho da população
r - fração da população a ser substituída por *cross-over*
m - taxa de mutação

Inicialize a população P: gere p soluções candidatas aleatoriamente
Avalie os elemento de P: para cada $x \in P$, calcule $f_i(x)$

Enquanto $\max\{f_i(x) : x \in P\} < f_t$, execute:
- Seja P' a nova população, iniciando com $P' = \emptyset$
- Selecione probabilisticamente $(1 - r)p$ elementos de P e adicione a P', sendo a probabilidade de $x \in P$ ser selecionado dada por:
  $$Pr(x) = f_i(x) / \sum_{x_j \in P} f_i(x_j)$$
- Probabilisticamente selecione $rp/2$ pares de elementos de P, de acordo com a distribuição dada por $Pr(x)$.
- Para cada par (x_i, x_j) execute *cross-over* e adicione os "filhos" a P'.
- Execute mutação de mp elementos de P'
- Faça $P \leftarrow P'$
- Calcule $f_i(x)$ para todo $x \in P$

4.2.6 Operador genético *cross-over*

Os operadores genéticos determinam como os membros de uma população são combinados, ou sofrem mutação, de forma a gerar novos candidatos. Na Fig. 4.1 é ilustrado o operador *cross-over* de um ponto, enquanto que o operador *cross-over* de dois pontos é exemplificado na Fig. 4.2. O operador de *cross-over* uniforme

consistem em utilizar uma máscara cujos bits são selecionados aleatoriamente com probabilidade uniforme. Já o operador de mutação simplesmente troca um ou mais bits de um cromossomo de 0 para 1 (ou vice-versa). Os bits são escolhidos aleatoriamente com probabilidade uniforme.

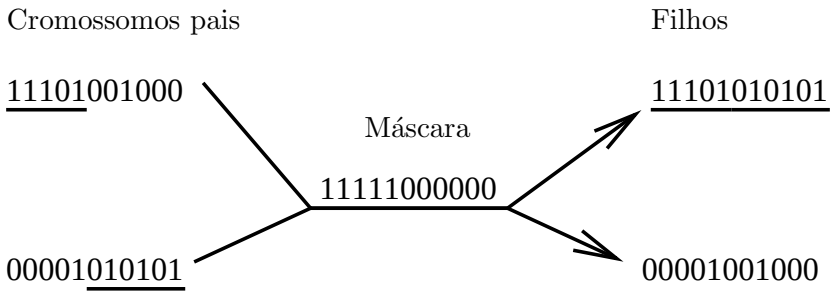

Figura 4.1: Operador *cross-over* de um ponto.

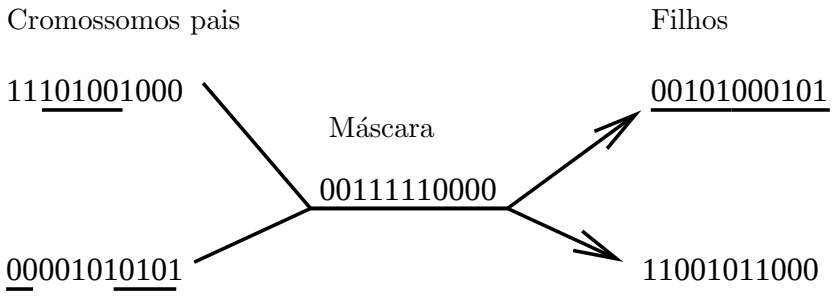

Figura 4.2: Operador *cross-over* de dois pontos.

4.2.7 Exemplo de aplicação

O problema consiste em encontrar o valor máximo da função $f(x) = x^2$ onde x pode assumir valores do conjunto $A = \{0, \ldots, 31\}$. Para aplicação do algoritmo genético, os seguintes passos são sugeridos:

- Utilize 5 bits para representar uma solução candidata.
- Defina a função de aptidão $f_i(x) = f(x)$.
- Gere um conjunto de 4 soluções candidatas randomicamente para compor a população inicial. (Tamanho da população $p = 4$).
- Adote o operador *cross-over* de um ponto como método de reprodução. (Taxa de substituição por *cross-over* $r = 50\%$).
- Adote como mecanismo de mutação a troca de um bit de uma solução candidata. (Taxa de mutação $m = 5\%$).

Na Tabela 4.1 estão indicados os valores iniciais armazenados na memória do algoritmo genético. Com esta população inicial e os parâmetros acima sugeridos, o algoritmo genético encontra a solução ótima para o problema: $x^* = 11111$ sendo $f(x^*) = 961$.

Tabela 4.1: População inicial

Solução Candidata	Fitness	Probabilidade de Reprodução
01101	169	14,4%
11000	576	49,2%
01000	64	5,5%
10011	361	30,9%

4.2.8 Questões práticas

O fenômeno de *crowding* ocorre quando um indivíduo com alta aptidão se reproduz rapidamente, gerando cópias ou filhos muito semelhantes que venham a constituir a maior parte da população. O aspecto negativo de *crowding* é a perda de diversidade, o que acarreta convergência prematura para ótimo local. O fenômeno de *crowding* é ilustrado na Fig. 4.3.

Algumas estratégias para evitar *crowding* são:

- seleção de pares para reprodução através de torneios em vez de seleção probabilística;
- utilizar o valor de posição (*ranking*) no lugar da aptidão para calcular as probabilidades de selecionar os indivíduos; e
- o valor de aptidão de um indivíduo pode ser reduzido pela presença de outros indivíduos semelhantes.

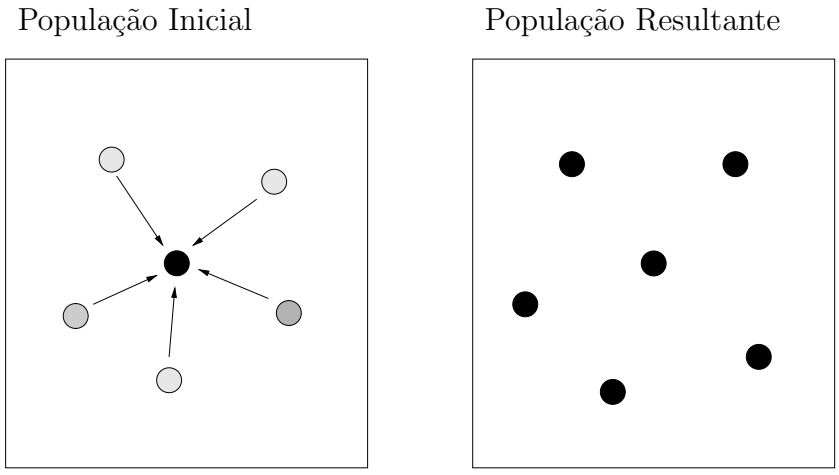

Figura 4.3: Fenômeno de *crowding*.

4.2.9 *Schema Theorem*

É possível descrever matematicamente a evolução da população de um AG em função do tempo? O *Schema Theorem* de Holland (1975) [7] oferece uma caracterização.

Definição 6 *Um Schema é uma cadeia de 0s, 1s e *s (curinga) que representa uma família de indivíduos.*

O *schema* "0*1" representa os indivíduos "001" e "011". *Schemas* podem ser utilizados para caracterizar a população de um algoritmo genético. Uma população de cadeias (vetores de bits 0s e 1s) pode ser vista em termos do conjunto de *schemas* que ela representa e o número de indivíduos associados com cada *schema*. Seja $m(s,t)$ o número de indivíduos no instante t (t-ésima população) que pertençam ao *schema* s. O *Schema Theorem* descreve o valor esperado[1] de $m(s,t+1)$ tomando como base: $m(s,t)$; as propriedades de s; as propriedades da população; e os parâmetros do AG (e.g., métodos de recombinação e taxa de mutação).

Teorema 1 *Considerando apenas seleção, tem-se*

$$E[m(s,t+1)] = \frac{u(s,t)}{f_m(t)} m(s,t)$$

onde: $m(s,t)$ é o número de indivíduos em P_t (t-ésima população) que pertençam ao schema s; $f_m(t)$ é o valor médio da função de aptidão dos elementos de P_t; e $u(s,t)$ é o valor médio da função de aptidão dos elementos de P_t que pertençam ao schema s.

Se vemos o AG como um processo virtual paralelo que:

i) executa uma busca através do espaço de possíveis *schemas* e, ao mesmo tempo,

ii) executa uma busca através do espaço de indivíduos, então a equação do Teorema 1 indica que *schemas* de maior aptidão crescerão em influência com o passar das gerações.

Teorema 2 *Considerando o operador cross-over de um ponto e mutação, tem-se*

$$E[m(s,t+1)] \geqslant \frac{u(s,t)}{f_m(t)} m(s,t) \left[1 - \frac{p_c d(s)}{l-1}\right] [1-p_m]^{o(s)}$$

onde:

- *p_c é a probabilidade do operador cross-over de um ponto ser aplicado a um indivíduo arbitrário;*
- *p_m é a probabilidade de que um bit arbitrário de um indivíduo arbitrário sofra mutação;*
- *l é o número de bits das cadeias;*
- *$o(s)$ é o número de bits definidos presentes no* schema *s (bits 0 ou 1); e*
- *$d(s)$ é a distância entre o bit definido mais à esquerda e o mais à direita no* schema *s.*

O Teorema 2 nos diz que os *schemas* de maior aptidão deverão aumentar sua influência, especialmente aqueles que possuem um pequeno número de bits definidos (contendo vários "*") e aqueles cujos bits definidos estão próximos uns dos outros.

[1] Denotado por $E[\bullet]$ e mais rigorosamente conhecido por esperança matemática

4.3 Referências

Os livros de Davis [4] e de Goldberg [6] trazem uma discussão ampla da filosofia por trás dos algoritmos genéticos, apresentam elementos teóricos e aplicações. Tais referências são indicadas para um aprofundamento na teoria e aplicações de algoritmos genéticos.

4.4 Exercícios

EX (O Problema de Bi-Partição Máxima em Grafos): Seja $G = (V, E)$ um grafo não direcionado cujas arestas possuem pesos não-negativos. Uma aresta $(u, v) \in E$ possui peso $w(u, v)$. Para dois conjuntos A e B de vértices, definimos $w(A, B)$ o peso total das arestas que têm um extremo em A e o outro, em B. Mais precisamente:

$$w(A, B) = \sum_{(u,v)\in E, u\in A, v\in B} w(u, v)$$

O problema é encontrar $S \subseteq V$ tal que $w(S, V - S)$ seja máximo.

Os dados de uma instância do problema da bi-partição máxima são fornecidos em arquivos tipo texto, cujo formato consiste do número de vértices, do número de arestas e de uma lista de arestas e pesos.

Exemplo de instância:
5 6
1 3 0,3
2 4 0,5
2 5 1,2
1 5 0,8
3 4 0,5
2 3 3,0

Tarefa: implemente um algoritmo Genético para resolver o problema de bi-partição máxima e teste o algoritmo em alguns problemas.

Capítulo 5

Considerações Finais

Este texto fez uma breve introdução ao campo da *inteligência computacional*, uma disciplina com raízes nas ciências cognitivas, filosofia, matemática e ciência da computação, que se preocupa com o estudo, projeto e implementação de agentes inteligentes. A inteligência computacional tem como propósito o entendimento dos princípios que levam ao comportamento inteligente. A diferença marcante da inteligência computacional e outras ciências que se preocupam com inteligência está na sua metodologia. A inteligência computacional busca construir artefatos que apresentam comportamento que seja considerado inteligente, independentemente desse comportamento ser obtido através da imitação de sistemas inteligentes biológicos ou através de outras teorias. Da mesma forma que máquinas são capazes de voar a partir das leis físicas e da aerodinâmica, sem necessariamente assemelhar-se a seres biológicos capazes de voar tais como pássaros, a preocupação é na síntese de artefatos inteligentes e não na reprodução de ocorrências naturais. O texto fez uma breve exposição de formalismos que podem ser utilizados, pelo menos em parte, na concepção de sistemas com comportamento inteligente.

As redes neurais (artificiais) são inspiradas nas redes neurais biológicas, as quais procuram capturar através das conexões da rede as relações sinápticas do cérebro humano, podendo ser reforçadas ou enfraquecidas. As redes neurais são amplamente empregadas na identificação de aproximações de uma função desconhecida, que se conhece pares de entrada e saída. Algoritmos eficientes do tipo *"backpropagation"* e variantes de funções de ativação (como funções de base radial) constituem um ferramental eficaz para resolução de vários problemas de treinamento de funções, identificação, classificação e controle de sistemas dinâmicos.

Conjuntos fuzzy generalizam o conceito clássico de conjuntos através de funções de pertinência. O conjunto das pessoas jovens, por exemplo, pode ser representado por um conjunto fuzzy. Para o universo das pessoas de 0 a 100 anos, um recém-nascido é certamente jovem, logo $\mu(0) = 1$, enquanto uma pessoa de 100 anos terá grau de pertinência $\mu(100) = 0$. Por outro lado, uma pessoa de 40 anos pode ser considerada mais ou menos jovem, assim poderíamos ter $\mu(40) = 0,5$. De maneira similar, a lógica fuzzy estendeu a lógica clássica (de dois valores, verdade e falsidade) para o domínio contínuo, onde as variáveis e proposições apresentam diferentes graus de verdade. As regras de combinação e inferência foram também modificadas para o caso fuzzy, procurando manter coerência e consistência com as regras da lógica

clássica. Por meio da lógica fuzzy, foi possível levar os sistemas especialistas clássicos para o domínio incerto, desenvolvendo os chamados sistemas especialistas fuzzy. Através de procedimentos de *"fuzificação"* (que transformam entradas precisas, em funções de pertinência) e de *"defuzificação"* (que transforma saídas fuzzy, em valores precisos), foi possível projetar controladores fuzzy que fazem uso de bases de regras. O texto fez uma introdução a conjuntos fuzzy e lógica fuzzy, finalizando com uma discussão sobre controladores fuzzy e apresentado exemplos didáticos.

Os algoritmos genéticos fazem parte da classe de métodos evolutivos, os métodos que se inspiram em processos naturais de evolução para resolver problemas diversos, sejam eles de engenharia, biologia ou matemática. Os algoritmos genéticos, por exemplo, partem do princípio da separação entre descrição e representação de soluções candidatas para um problema particular. As soluções candidatas são codificadas através de cadeias de bits, de forma assemelhada à codificação do DNA, e através de um processo de seleção externo às soluções evoluem de uma geração para outra, aumentando a qualidade dos membros da população a cada geração. A seleção se dá com base em uma função de aptidão (*"fitness function"*), a qual avalia a qualidade da solução candidata codificada por uma cadeia de símbolos. Inúmeras são as aplicações de algoritmos genéticos e existem vários outros métodos evolutivos, como programação genética, colônia de formigas e *swarm intelligence*.

Referências Bibliográficas

[1] A. Church. *Introduction to mathematical logic*. Princeton University Press, 1956.

[2] E. Cox. Fuzzy fundamentals. *IEEE Spectrum*, pages 58–61, October 1992.

[3] C. Darwin. On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life. 1859.

[4] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[5] G. Goebel. An introduction to fuzzy control systems. June 2003.

[6] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, 1989.

[7] J. Holland. *Adaptation in Natural and Artificial Systems*. Michigan University Press, 1975.

[8] J. Jantzen. Tutorial on fuzzy logic. Technical Report 98-E 868, Department of Automation, Technical University of Denmark, August 1998.

[9] E. H. Mamdani. Application of fuzzy logic to approximate reasoning using linguistic synthesis. *IEEE Transactions on Computers*, C-26(12):1182–1191, 1977.

[10] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.

[11] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer-Verlag, 1999.

[12] G. Orr, N. Schraudolph, and F. Cummins. Neural networks. Lecture Notes for CS-449, Willamette University, Fall 1999.

[13] D. Poole, A. Mackworth, and R. Goebel. *Computational Intelligence: A Logical Approach*. Oxford University Press, 1998.

[14] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, 1st edition, 1995.

[15] K. Self. Designing with fuzzy logic. *IEEE Spectrum*, pages 42–44, 105, November 1990.

[16] R. Tanscheit. Sistemas fuzzy. Minicurso, VI Simpósio Brasileiro de Automação Inteligente, Bauru, SP, Setembro 2003.

[17] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[18] L. A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man, and Cybernetics*, 1:28–44, 1973.

[19] L. A. Zadeh. The concept of linguistic variable and its applications to approximate reasoning. *Information Sciences*, 8:43–80, 1975.

[20] H.-J. Zimmermann, editor. *Fuzzy set theory and its applications*. Kluwer, Boston, 1st edition, 1991.