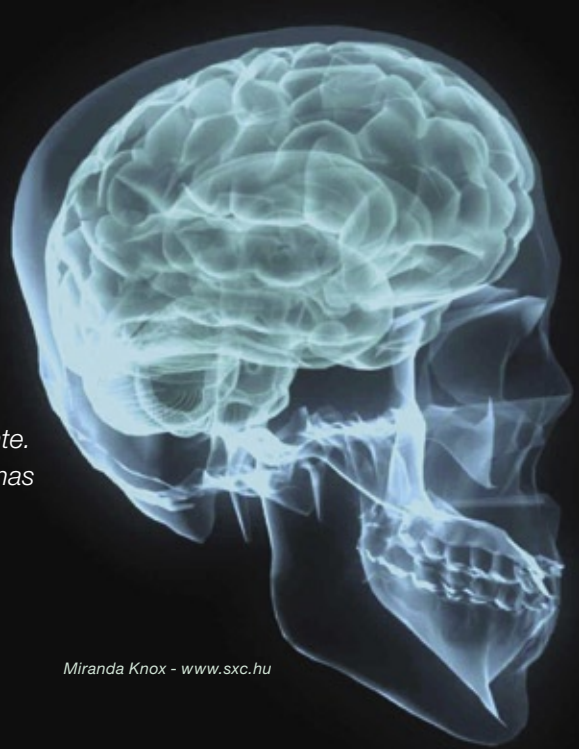
Programação de redes neurais com a Libfann

Jogos cerebrais



Miranda Knox - www.sxc.hu

3, 4, 8, 11… ? Uma rede neural consegue completar essa seqüência sem conhecer seu algoritmo subjacente. Veja como as redes neurais ajudam a resolver problemas simulando o comportamento de um cérebro.

por Andreas Romeyke

Ao procurar um caminho num mapa, seus olhos cairão diretamente numa solução eficiente. O cérebro humano é capaz de fazer julgamentos sem muita atenção a algoritmos de otimização ou cálculos de distância. Essa técnica intuitiva é totalmente alheia aos computadores digitais. Os programas de computador convencionais tendem a operar através de soluções matemáticas, tornando-os ineficientes em tarefas como predição e reconhecimento de padrões. Uma forma experimental de programa conhecida como *Rede Neural Artificial* (RNA, ou *ANN* na abreviação em inglês) resolve esse problema fazendo o computador funcionar de forma mais semelhante a um cérebro biológico.

Uma rede neural artificial simula um conjunto de células nervosas conectadas por caminhos ponderados. Um uso de sucesso para redes neurais é o campo do reconhecimento de faces. Uma rede neural consegue reconhecer um rosto com base num conjunto de pixels coloridos, apesar de ruído ou distorção, exatamente como um humano. Outras aplicações para a tecnologia de redes neurais incluem o reconhecimento óptico de caracteres ou previsões de manchas solares e preços de ações.

Neste artigo, veremos os princípios básicos das redes neurais, e introduziremos a biblioteca *Libfann*[1], que pode ser usada para criação de aplicações com uso de redes neurais.

Modelo natural

Uma rede neural simula a estrutura de um cérebro. Ela modela o efeito de um conjunto de neurônios que influenciam os estados uns dos outros através de um grande número de conexões. O peso diferencial das conexões neurais, que representam as fibras nervosas do cérebro, produz um valor de saída específico para um padrão específico de neurônios receptores. As conexões entre neurônios são ajustadas através de um processo análogo a um *treinamento*. Nesse processo, a rede neural aprende a associar padrões de entrada específicos a valores de saída específicos. Se o treinamento tiver sucesso, o cérebro artificial será capaz de descobrir soluções não especificamente apresentadas como exemplos.

A **figura 1** mostra uma célula nervosa, o modelo natural dos neurônios de uma RNA. A célula nervosa contém o corpo e dendritos que saem deste. Os dendritos transportam impulsos elétricos ao corpo da célula. Se a soma total desses impulsos exceder um valor limite pré-definido (potencial de ação), o neurônio torna-se ativo, enviando impulsos às células às quais está conectado.

Um neurônio artificial simula as propriedades de seu equivalente

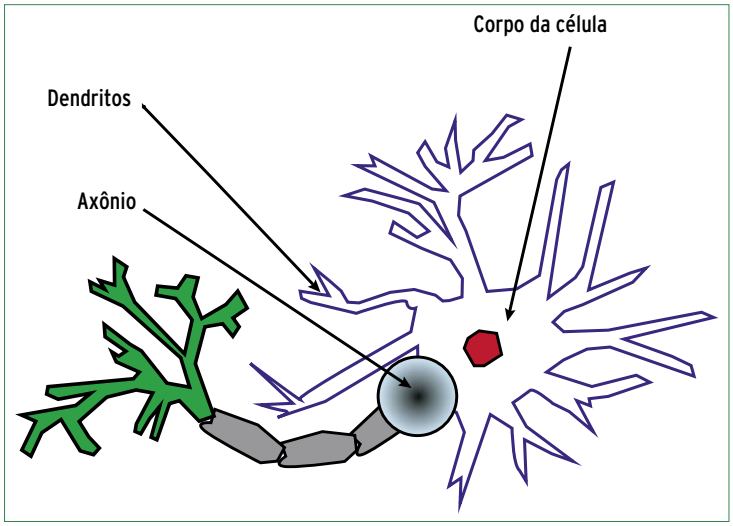


Figura 1 Os neurocientistas consideram as ramificações das células nervosas como a base do poder do cérebro para reconhecer padrões ou prever estados do sistema difíceis de calcular.

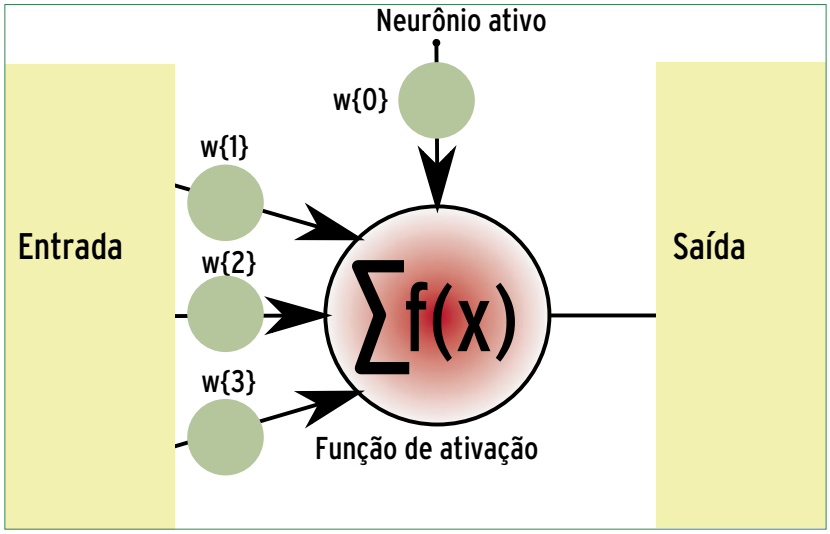


Figura 2 Assim como seus equivalentes biológicos, neurônios artificiais calculam a soma do potencial de ação dos neurônios conectados a eles, e repassam o sinal para outros neurônios através de conexões ponderadas.

natural: ele soma os potenciais de seus dendritos, aplica uma função fixa especial de ativação e passa os resultados a todas as células conectadas a ele (**figura 2**). As conexões com outros neurônios são ponderadas para atenuar ou amplificar o sinal ao longo de seu caminho.

A *função de ativação* define o limite no qual o neurônio será ativado. Abaixo desse valor, o neurônio não enviará sinais. Essa função freqüentemente é uma função simples de limite que retorna 1 se a soma de todas as saídas for superior a um valor específico. É comum representar a função de ativação num neurônio separado conhecido como *ativar neurônio* (*on neuron*). Então, pode-se ponderar a *ativar neurônio* como as conexões a outros neurônios.

Projeto

O processo de treinamento adapta a rede neural a uma situação específica; porém, na escala estrutural, o desenvolvedor também pode escolher uma topologia para a rede neural que reflita seu uso pretendido. Diferentes tipos de conexões entre neurônios levam a redes com características diferentes[2].

Uma das topologias de rede mais simples, bem explorada pela pesquisa científica, é o modelo de perceptron multicamada (*MLP*, na sigla em português) pró-alimentado[2]. Esse modelo divide a rede em camadas separadas. Essa rede não possui *feedback*; em outras palavras, o potencial de atuação simplesmente se propaga da esquerda para a direita (veja a **figura 3**).

As capacidades de uma rede neural, tais como a habilidade de reconhecer padrões ou prever valores, são um produto da estrutura interna da rede.

As operações a seguir modificam as características de uma RNA:

- adicionar novas conexões ou eliminar as já existentes;
- modificar os pesos das conexões entre neurônios;
- modificar os valores de limite dos neurônios;
- acrescentar ou eliminar neurônios.

O treinamento oferece os pesos adequados para resolver um problema específico. No caso do reconhecimento de caracteres, a entrada seria uma imagem ou um trecho de um texto e os respectivos códigos de caracteres. No caso do comportamento do mercado de ações ou da atividade das manchas solares, os dados históricos são usados para treinar a rede neural (**figura 4**). Uma função de aprendizado compara a entrada do exemplo com os valores de objetivo, e modifica as conexões neuronais, pesando até a reação da rede ser correspondente ao objetivo.

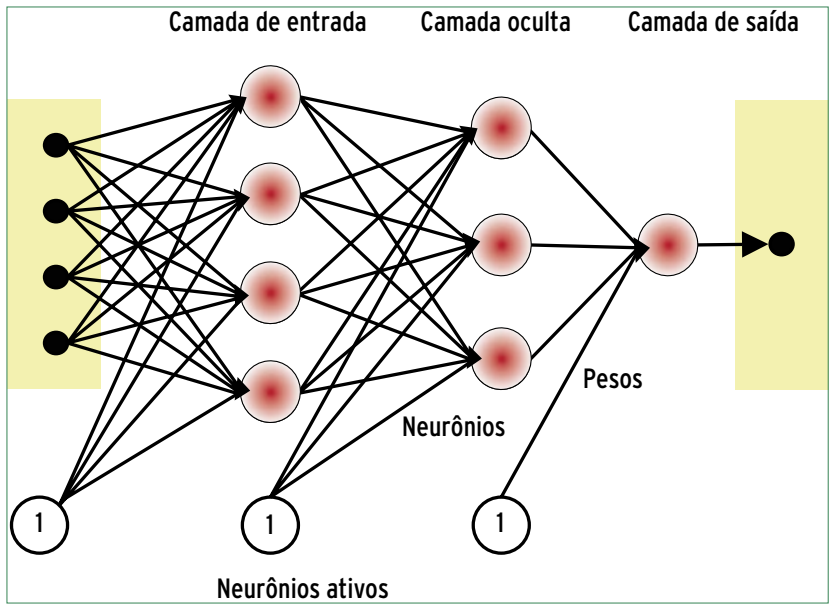


Figura 3 O perceptron multi-camada, que permite a propagação do potencial da entrada para a saída sem *loops* de retorno, é a rede neural artificial mais simples e melhor explorada.

Dentro da mente

Descreveremos agora, com um exemplo simples, o que acontece na fase de aprendizado de uma rede neural. Imagine que queiramos uma rede com quatro neurônios para prever o valor médio de dois números (veja a **figura 5**). No lado esquerdo da figura, os números `0,1` e `0,3` são entrados nos neurônios receptores. Inicialmente, as conexões neuronais possuem pesos aleatórios. A função de ativação, que define a forma como um neurônio reage à entrada, é $f(x) = x$. RNAs mais poderosas precisam de funções mais complexas, obviamente, mas esse exemplo simples é adequado para explicar o princípio subjacente.

Se os neurônios receptores possuem os valores de 0,1 e 0,3, pesar as conexões para os potenciais oferecem os seguintes valores: $(0{,}1*1{,}0 + 0{,}3*0{,}9 + 1{,}0*0{,}4) = 0{,}77$ para o primeiro neurônio $N(1,1)$, e $(0{,}1*0{,}2 + 0{,}3*0{,}3 + 1{,}0*0{,}7) = 0{,}81$ para o segundo neurônio $N(1,2)$. O neurônio entre as camadas de entrada e saída possui valor de $(0{,}77*0{,}5 + 0{,}81*0{,}1 + 1{,}0*0{,}2) = 0{,}666$. O neurônio de saída retorna o valor de $(0{,}666*0{,}2 + 1{,}0*0{,}3) = 0{,}433$, embora a média correta dos números 0,1 e 0,3 seja 0,2.

Em outras palavras, a rede não chegou muito perto do valor correto na primeira tentativa. Para permitir que a RNA ajuste sua matemática, é necessário modificar a pesagem das conexões neuronais. A contribuição do erro permite descobrir qual pesagem entre quais neurônios devem ser corrigidas. A contribuição do erro é o quadrado do valor de saída esperado menos o quadrado dos valores retornados nos neurônios de saída. O valor resultante é conhecido como erro quadrado médio (*MSE* ou *MQLE*, na notação em inglês).

Marcha a ré

Potenciais de ação geralmente movem-se adiante na rede a partir da entrada em direção à saída (sentido normal). Um método de ensino conhecido como propagação reversa (ou *back propagation*, como costuma ser chamado) inverte essa direção. Ele informa o valor de erro retornado na saída de volta, através da rede em direção à entrada, com base nos pesos das conexões individuais. A distribuição dos valores de erro sobre os nós da rede oferece a base para a modificação dos pesos. Os especialistas já desenvolveram vários outros métodos de ensino além desse, e alguns prometem resultados melhores para certas tarefas.

A **figura 6** mostra como a contribuição do erro se propaga para trás a partir da saída. O potencial do neurônio de saída é a soma de suas duas conexões: aquela com o neurônio ativo, com peso de 0,3, e a com o neurônio na camada abaixo, com peso de 0,2. Com base nisso, a contribuição do erro $(0{,}433 - 0{,}2 = 0{,}233)$ no neurônio de saída é distribuída pelas duas conexões. O caminho para o neurônio ativo tem uma fatia de $0{,}3/(0{,}2+0{,}3) = 60\%$, e o caminho para o neurônio abaixo tem a fatia de $0{,}2/(0{,}2+0{,}3) = 40\%$. Essa técnica possibilita o cálculo do potencial de erro total para cada conexão neuronal.

Ao final, um fator fixo de aprendizado estipula como uma contribuição de erro influencia o peso. Uma boa escolha de fator de aprendizado é um pré-requisito importante para o treinamento eficaz. Assim como vários outros parâmetros da rede, esse fator é desconhecido no início do treinamento. O treinamento completo de uma RNA sempre envolve um grande número de ciclos de propagações reversas, com pares de valores de entrada e saída para o problema que deve ser resolvido pela rede ao final.

Plano de treinamento

É óbvio que os pesos jamais devem ter valor zero, pois não haveria forma de traçar os erros. Pesos muito semelhantes ou com diferenças muito grandes teriam efeito negativo sobre o processo de aprendizado. Para uma RNA eficiente, é preferível que os sinais se propaguem através de toda a rede, exceto por áreas específicas, para lidar com padrões específicos.

Em aplicações práticas, a simples função de ativação $f(x) = x$ será substituída por uma tangente hiperbólica ou uma função sigmoidal. Isso aumenta o desempenho da rede neural, pois essas funções podem mapear

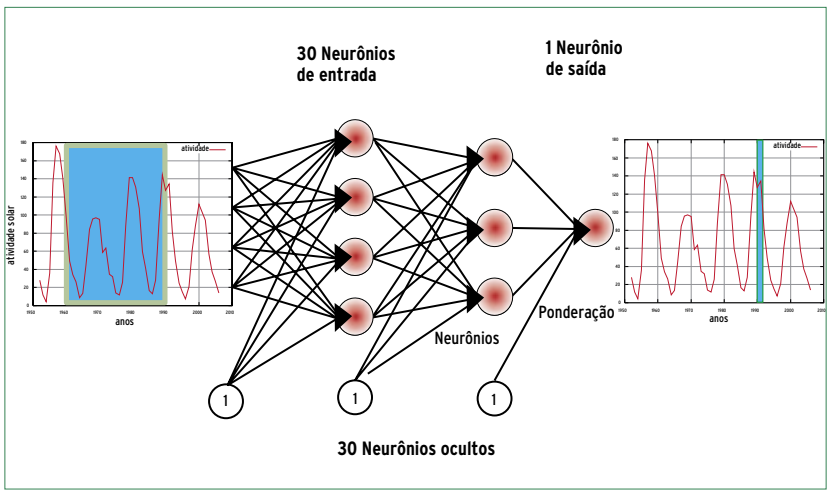


Figura 4 Uma rede neural com três camadas de neurônios prevê a atividade das manchas solares nos últimos 30 anos, e assim aprende a prever o fenômeno para o ano que vem.

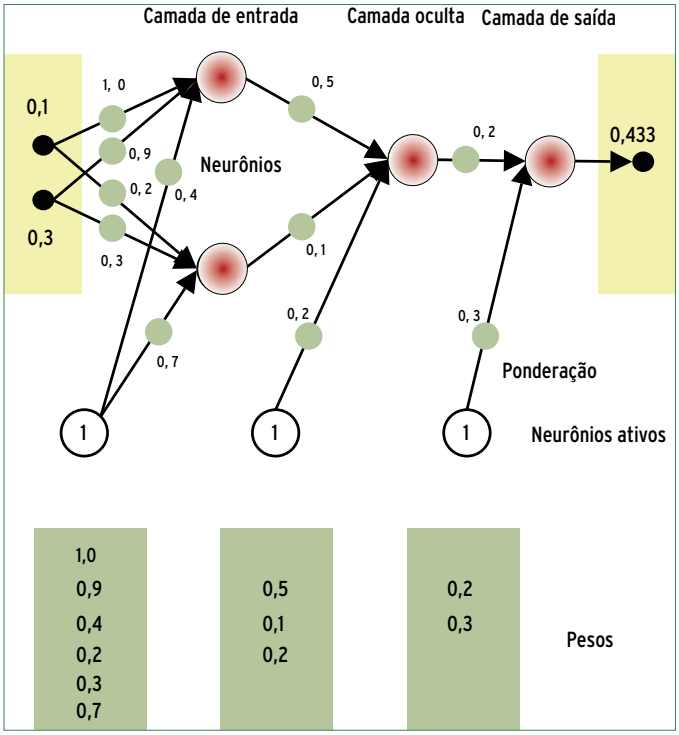


Figura 5 O comportamento da rede neural é definido pela ponderação das conexões neuronais e os neurônios ativos, que fixam o limite pelos quais os neurônios passarão estímulos para os outros.

uma faixa de entrada até o infinito com valores úteis. A back propagation também pressupõe que a função de ativação possa ser invertida. A vantagem desse esforço adicional é que os perceptrons de três camadas são capazes de aprender funções matemáticas arbitrárias, supondo que usem funções de ativação não lineares adequadas.

Libfann

A biblioteca *Fast Artificial Neural Network* (*FANN*) é uma biblioteca gratuita e de Código Aberto que fornece uma interface em C para implementar redes neurais multicamadas. A biblioteca foi desenvolvida em 2003 por Steffen Nissen, na Universidade de Copenhague, como parte de sua pesquisa científica, e ainda está em desenvolvimento ativo. A Libfann é fácil de usar e bem documentada, e roda em qualquer plataforma popular. A página do projeto também possui alguns exemplos práticos que facilitam a introdução a seu uso. Fora C, há ligações para todas as linguagens de programação mais comuns. No momento da escrita deste artigo, a Libfann é uma das implementações mais rápidas para simulações de redes neurais.

A maioria das distribuições Linux inclui a versão 1.2 da biblioteca. Um `aptitude install libfann1-dev` instala-a em qualquer distribuição derivada do *Debian*. O código-fonte, que pode ser compilado com os passos tradicionais, `configure; make; make install`, está disponível em [1].

Compilação

Todas as aplicações de redes neurais são diferentes, e é impossível explorar todas as sutilezas desse complexo campo num único artigo. O site do projeto Libfann inclui um manual de referência com descrições e notas de uso para as funções da biblioteca. O site da **Linux Magazine** contém em [3] um exemplo de programa em C que cria uma rede neural e a treina.

Para os que desejarem experimentar, algumas das funções mais importantes da Libfann são `fann_train()`, `fann_run()` e `fann_test()`. A função `fann_train()` espera uma *struct* de rede, `struct fann * ann;`, como seu primeiro parâmetro. Pode-se chamar `ann=fann_create(taxa_conexão, taxa_aprendizado, num_camadas, num_entrada, num_neurônios_ocultos, num_saída);` para criar a struct. `taxa_conexão` especifica a força das conexões entre os neurônios. O valor correto normalmente é `1.0`. `taxa_aprendizado` deve ficar entre `0.7` e `0.00001`. O parâmetro `num_camadas` e os valores após ele informam à Libfann o número de camadas da rede e o número de neurônios em cada uma delas.

Treino e pensamento

Paralelos com o pensamento humano são úteis para entender o que se passa durante o treinamento e descobrir a fonte de quaisquer problemas numa rede neural – afinal, elas emulam a estrutura do cérebro. Se a sessão de treinamento informar os dados históricos em ordem cronológica, a rede talvez desenvolva visão em túnel. Isso afetaria a capacidade da RNA de lidar com novos dados.

Uma ordem aleatória evita a generalização prematura, e portanto a necessidade de treinar novamente a rede neural após uma estrutura inválida ser estabelecida nas conexões neuronais. Por isso, o script em *Perl*[3] garante uma ordem aleatória aos dados.

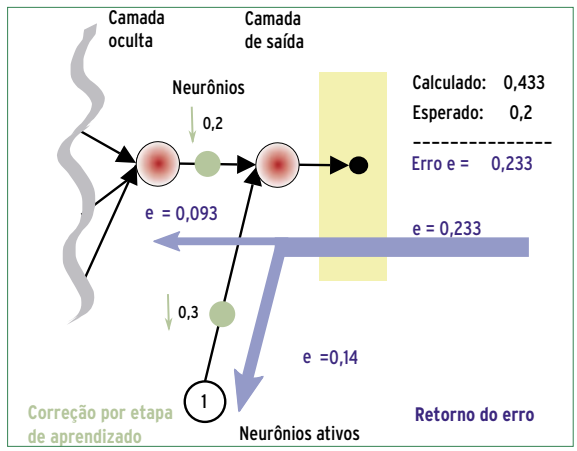


Figura 6 Redes neurais aprendem cometendo erros de previsão de valores específicos, rastreando esses erros de volta por sua estrutura e re-pesando as conexões entre neurônios que contribuíram para esses erros.

Dados que a rede não tenha visto durante o treinamento ajudam a julgar a eficácia com que a RNA consegue lidar com abstrações no estado atual do treinamento. O script em Perl divide os dados em dois subconjuntos. O erro que ocorre nesse caso é chamado de erro médio quadrado de generalização, ou *MQGE*. Junto com o *MQLE* (erro médio quadrado de aprendizado), ele informa se a rede neural está pronta para prever o futuro ou se é necessário mais treinamento. A Libfann injeta os dois valores através das funções `fanntest(rna, vetor_entrada, vetor_saídaesperada)` e `fann_get_MSE()`. Por último, `fann_save(rna,arquivo)` armazena a estrutura da rede e os pesos atuais para uso futuro.

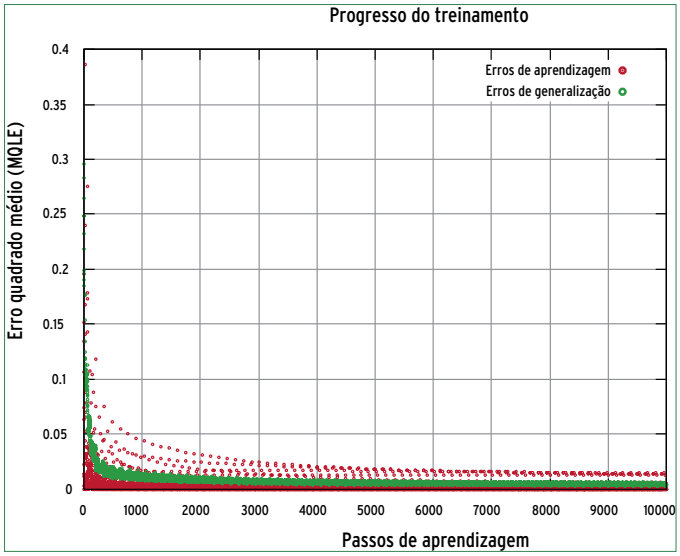


Figura 7 Num treinamento de sucesso, o *MQLE* cai continuamente, e a rede vai melhorando seu desempenho.

Vida real

O sucesso do treinamento depende não apenas dos dados, mas também da adequação da estrutura da rede à tarefa em questão – a começar pela função de ativação.

A Libfann usa a função sigmóide por padrão, e isso não é um problema para prever a atividade das manchas solares e outros fenômenos que se encaixam numa faixa positiva. Porém, no caso de preços de ações e outras séries temporais com valores negativos, é necessária a função de tangente hiperbólica `fann_set_activation_function_output(rna, FANN_SIGMOID)`.

O fator de aprendizado também influencia fortemente o sucesso ou fracasso do treinamento, pois especifica qual o efeito dos erros de aprendizado sobre os pesos das conexões entre neurônios, e sobre o número de neurônios da rede. O número de neurônios na camada intermediária deveria manter-se num mínimo, inicialmente. Três ou um máximo de 15 neurônios são suficientes para a maioria das aplicações. A tentativa e erro também oferecerão uma medida dos números adequados. Para essa sessão de treinamento, 500 mil passos de aprendizado devem ser um número suficiente.

Se o número de erros de aprendizado não cair continuamente, a rede fica parada num mínimo local, e seu desempenho dificilmente melhora, não importa a duração do treinamento. Nesse caso, é necessário reiniciar o treinamento com um fator de aprendizado menor, e possivelmente alterando a estrutura da rede.

Inspecionar o arquivo `fann_save` pode revelar o motivo do baixo desempenho da rede simplesmente pelo treinamento: neurônios individuais com pesos excessivos freqüentemente interferem no processo de aprendizado.

Se a curva de aprendizado continuar caindo, como mostrado na **figura 7**, então é hora de consultar o erro de generalização: se a curva for suave, não se deve esperar grande capacidade de previsão. A rede aprendeu os valores de treinamento e receberá valores de entrada desconhecidos. Para alterar isso, é necessário reduzir o número de neurônios ocultos.

Se o erro de generalização estiver num nível consistentemente alto, o número de nós ocultos está pequeno demais, ou a sessão de treinamento não foi suficientemente intensa.

A função `fann_load` da Libfann carrega uma rede armazenada anteriormente com `fann_save()`. A função `fann_run(rna, entrada)` retorna a saída da rede treinada. O script em Perl automatiza o teste do cérebro artificial. Nesse teste, a saída de uma rede treinada com sucesso foi bastante próximo às previsões.

Conclusão

A Libfann facilita a criação, treinamento e uso de RNAs. Os usuários não precisam preocupar-se com detalhes matemáticos como a inversão da função de ativação.

Escolher parâmetros como a taxa de aprendizado e o número de neurônios intermediários realmente demanda experiência e paciência. Os erros de aprendizado e generalização, além de um conhecimento da saturação de neurônios individuais, fornecem indicações dos motivos de falha de uma rede específica. A Libfann ajuda a precisar esses valores.

A versão atual (2.0) da Libfann estende seu escopo funcional, acrescentando novos algoritmos de aprendizado e tipos de neurônios. ■

Mais informações

[1] Libfann: http://fann.sourceforge.net

[2] Redes neurais artificiais: http://pt.wikipedia.org/wiki/Rede_neural

[3] Programa de treinamento: http://www.linuxmagazine.com.br/

[4] Warren Sarle, FAQ sobre RNAs: http://www.faqs.org/faqs/ai-faq/neural-nets/