UNIVERSIDADE DE PERNAMBUCO
ESCOLA POLITÉCNICA DE PERNAMBUCO

Disciplina: Inteligência Artificial
Prof. Dr. Ruben Carlo Benante

Período: 2019.1
Data: 26/06/2019

Backpropagation

Hugo Branco Wanderley Barbosa

1 INTRODUÇÃO

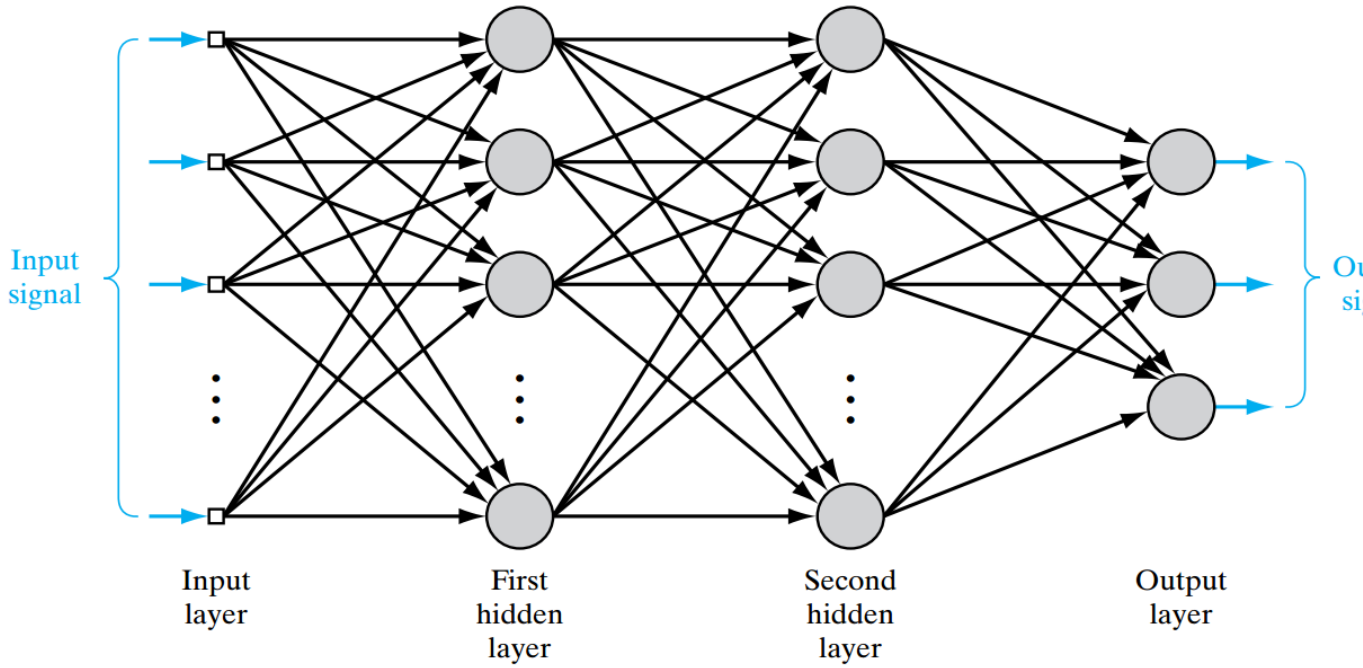
O objetivo deste trabalho foi estudar, compreender e reproduzir uma rede neural cujo processo de aprendizado utilizado fosse o *Multilayer Backpropagation*. Para entender o conceito desse processo de aprendizado, partimos de uma estrutura essencial para o funcionamento de um cérebro: o neurônio. Nas redes neurais, os equivalentes dos neurônios são chamados de perceptrons e embora seu mecanismo seja diferente de um neurônio verdadeiro seu objetivo permanece o mesmo: realizar pequenas tarefas a fim de transmitir uma informação de neurônio em neurônio (neste caso, perceptron), num processo de cadeia, até que a informação útil seja processada e transmitida para seu destino.

Na *Multilayer Backpropagation*, a primeira metade de seu nome resulta da estruturação de seus perceptrons na rede neural, que tem como consequência os pontos tanto positivos quanto negativos de seu funcionamento. Diz-se que a rede neural possui múltiplas camadas pois o processamento das informações através dos perceptrons pode ser dividido em etapas; em cada uma destas etapas, um certo número de perceptrons escolhido pelo programador recebe informações da camada anterior, executa cálculos com as mesmas e as repassa para uma próxima etapa do processo de aprendizado. O sistema de múltiplas camadas tem como vantagem permitir que a rede neural construída possa distinguir três ou mais padrões de dados de entrada, ao passo que sua desvantagem, também causada pela estruturação multi camadas, é a dificuldade de transformar seus perceptrons a fim de relacionarem melhor os dados de entrada a seus respectivos valores de saída devido a alta conectividade entre os perceptrons.

O termo *backpropagation* tem origem na forma da qual a rede se transforma a fim de criar ou melhorar a relação entre os dados de entrada e a saída da informação processada. Primeiro, na fase de progresso (termo livremente traduzido à partir da expressão *foward phase*), os pesos dos perceptrons são fixos e as informações de entrada são captadas por todos perceptrons da primeira camada, processadas por uma função diferenciável em cada um deles resultando num subproduto para, então, ser processador pela camada de perceptrons seguinte até que o resultado final seja processado.

Na segunda fase, chamada fase de retorno (ou *backward phase*), um sinal de erro é gerado ao comparar o resultado processado com o resultado esperado. O sinal passa por funções (derivadas das funções da fase de progresso) para que uma correção possa ser propagada de volta (*backpropagated*) por toda rede neural a fim de modificar os pesos de cada perceptron, melhorando o processo de classificação da rede.

Figura 1 – Exemplo de uma rede neural construída à partir de *backpropagation*. Imagem retirada do Haykin (2009).



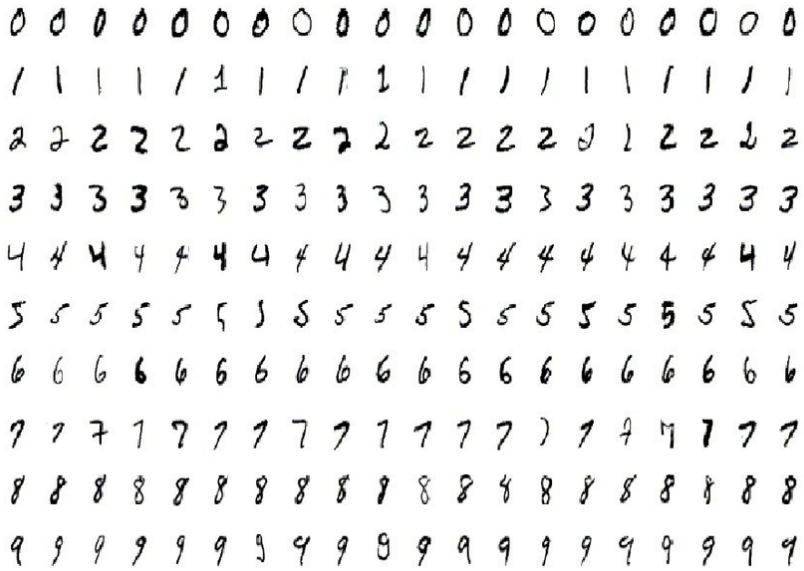
Conforme visto na figura, as camadas podem ser classificadas como invisíveis (*hidden*) ou de saída (*output*). A camada de saída é a última camada de perceptrons em que eles recebem valores da penúltima camada, realizam cálculos utilizando a função da camada e então retorna um resultado final que classifica os dados de entrada; já as camadas invisíveis são responsáveis por captar os dados de entrada e transformá-los em valores intermediário a serem utilizados pelas próximas camadas, até a última. As camadas invisíveis reconhecem características que diferenciam uma entrada da outra enquanto a última camada, a de saída, reúne todas características da entrada e fim de nos retornar uma classificação final.

Para obter a melhor rede neural possível, o erro final deve ser mínimo. Sabendo que o erro é uma função diferenciável dos dados de entrada, podemos encontrar seu valor mínimo à partir de sua primeira derivada igualada a zero.

A base de dados utilizada neste projeto, a *MNIST database* (*Modified National Institute of Standards and Technology database*), consiste em números de 0 a 9 manuscritos e digitalizados de forma que encaixe-se num quadrado com 28 por 28 pixels; cada pixel tem um valor entre 0 e 255 para determinar sua cor que vai de branco a preto absoluto e todas as imagens são

acompanhadas de uma “legenda” para cada algarismo escrito. O tamanho da base consiste em 60 mil imagens para treinamento e 10 mil imagens para testes, porém este trabalho utilizou apenas 6 mil imagens para o treinamento e 4 mil imagens para os testes.

Figura 2 – Exemplo de alguns algarismos manuscritos encontrados no banco de dados MNIST.



2 ALGORITMO

Ao iniciar o planejamento da rede neural, define-se a quantidade de camadas de perceptrons desejada assim como o total de perceptrons em cada uma delas. É necessário que uma camada possua seu número de perceptrons menor ou igual ao número de perceptrons na camada anterior, considerando que a ordenação inicia na etapa da entrada de dados e conclui-se na saída da classificação. Um número menor de perceptrons na camada seguinte significa que algumas informações são unidas a outras e sua divisão posterior torna-se impossível devido à arquitetura da rede neural; a escolha de ter uma camada com mais perceptrons que sua camada anterior só seria justificada se a camada em questão possuísse mais informações que sua antecedente, o que não acontece. Nesse caso trabalhado, escolheu-se, de forma arbitrária, que trabalharíamos com três camadas de perceptrons, sendo duas invisíveis e uma de saída, e o número de perceptrons em cada camada seria 300, 100 e 10, sendo a última aquela de saída.

A saída de dados, um vetor de dez números, nos informa a probabilidade do número de entrada pertencer a cada uma das dez classificações (números de zero a nove). Portanto, por exemplo, o vetor $a = \{0,\ 0.3,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0.7,\ 0,\ 0\}$ nos informa que a entrada tem 70% de chances de ser o algarismo 7 e 30% de chances de ser o algarismo “1” conforme uma rede neural arbitrária previamente construída.

2.1 Foward Phase

Conforme explicado, cada imagem consiste na entrada de 784 pixels, formatados em números inteiros entre 0 e 255. A primeira camada de perceptrons recebe os valores e efetua seus cálculos segundo demonstrado na equação (1). Considere y_i^o o valor de cada pixel de entrada, onde i é o número total de pixels da imagem e j é o número total de perceptrons na primeira camada. A matriz w_{ji} é a matriz de pesos de cada perceptron com valores inicialmente aleatórios, entre -0.5 e 0.5. A magnitude de cada peso pode nos informar a importância de cada dado singular para a classificação final dos dados. O termo b_i é o chamado vetor *bias*. O total de imagens utilizadas para o treinamento é de $n = 6000$.

$$v_j^{(1)}(n) = \sum_{i=0}^{784} w_{ji}^{(1)}(n)\ \ y_i^{(o)}(n) + b_j^{(1)} \tag{1}$$

Portanto, cada perceptron j calculará um valor para v_j baseado nos dados de entrada e *bias*. Aplica-se sobre este valor v_j uma função de ativação; são diversas as funções que podem ser utilizadas para a ativação, como função linear, hiperbólica ou binária (*hard limit*). Nesse trabalho utilizou-se a função de ativação sigmóide, conforme demonstra-se na equação (2).

$$\begin{aligned} y_j^{(1)}(n) &= \varphi(v_j^{(1)}(n)) \\ \text{onde}\ \ \varphi(x) &= \frac{1}{1+e^{-x}} \\ \therefore\ \ y_j^{(1)}(n) &= \frac{1}{1+e^{-v_j^{(1)}(n)}} \end{aligned} \tag{2}$$

Em resumo: até então cada perceptron na primeira camada recolhe valores de todos 784 pixels da imagem e transforma esse vetor num escalar. Como na primeira camada temos 16 perceptrons trabalhado, obtemos como produto intermediário um vetor de 16 elementos que servirá como entrada para a segunda camada, também invisível. As equações são as mesmas:

$$\begin{aligned} v_j^{(2)}(n) &= \sum_{i=0}^{16} w_{ji}^{(2)}(n)\, y_i^{(1)}(n) + b_j^{(2)} \\ y_j^{(2)}(n) &= \varphi(v_j^{(2)}(n)) \end{aligned}$$

Prosseguimos para a terceira camada de perceptrons que recebe um vetor de $i = 16$ elementos e retorna um vetor com $j = 10$ elementos:

$$\begin{aligned} v_j^{(3)}(n) &= \sum_{i=0}^{16} w_{ji}^{(3)}(n)\, y_i^{(2)}(n) + b_j^{(3)} \\ y_j^{(3)}(n) &= \varphi(v_j^{(3)}(n)) \end{aligned} \tag{3}$$

2.2 Backward Phase

Na posse do resultado dado pela rede neural, compara-se o vetor saída com o vetor que era esperado da rede, com valores que pudessem descrever exatamente o algarismo manuscrito. Sendo assim, espera-se um vetor com valores majoritariamente nulos cuja exceção seja um elemento de valor unitário na posição equivalente ao número real.

Tabela 1 – Exemplo de saída da rede em comparação ao seu respectivo exemplo de resultado ideal (neste caso, o algarismo "8").

Esperado	0	0	0	0	0	0	0	0	1	0
Obtido	0.10	0.13	0.03	0.65	0.05	0.00	0.20	0.11	0.88	0.35

A função erro é medida através da função dos mínimos quadrados. Considere na equação (4) $R_i(n)$ o resultado esperado para cada um dos dez elementos no vetor final. A fim de minimizar a função erro, podemos igualar sua derivada a zero mantendo em mente que a função depende de outras funções, portanto utiliza-se a regra da cadeia conforme demonstrado na equação (5), que aplica a regra da cadeia, do cálculo diferencial, em nossa rede neural na última de suas camadas: a camada de saída (*output layer*). As derivadas em questão são efetuadas em relação à matriz de pesos pois são as únicas variáveis nas funções; não podemos alterar os valores de entrada ou saída.

$$E(n) = \sum_{i=0}^{9} \Big(R_i(n) - y_i^{(3)}(n) \Big)^2 \tag{4}$$

$$\frac{\delta E(n)}{\delta w_{ji}^{(3)}} = \frac{\delta E(n)}{\delta v_j^{(3)}} \frac{\delta v_j^{(3)}}{\delta w_{ji}^{(3)}} \tag{5}$$

Considerando a equação (5), é possível obter as derivadas parciais aplicando a regra da cadeia. A segunda derivada parcial apresentada, $\delta v_j^{(3)} / \delta w_{ji}^{(3)}$, é resolvida aplicando-se a regra da cadeia visto que um termo é função do outro, como vimos em (3). Então, obtemos $\delta v_j^{(3)} / \delta w_{ji}^{(3)} = \dot{v}_j^{(3)} = y_i^{(2)}(n)$. Para o *bias*, $\delta v_j^{(3)} / \delta b_j^{(3)} = 1$. Quanto à primeira derivada, $\delta E(n) / \delta v_j^{(3)}$, ela nos fornece uma matriz de valores que chamamos de **gradientes locais** ou **sensitividades**. A derivada parcial em questão é resolvida conforme segue:

$$\frac{\delta E(n)}{\delta v_j^{(3)}} = s_i^{(3)}$$

$$s_i^{(3)} = \frac{\delta \left(R_i(n) - y_i^{(3)}(n)\right)^2}{\delta v_j^{(3)}}$$

$$s_i^{(3)} = 2 \left(\cancelto{0}{\frac{\delta\ R_i(n)}{\delta v_j^{(3)}}} - \frac{\delta\ y_i^{(3)}(n)}{\delta v_j^{(3)}} \right) \left(R_i(n) - y_i^{(3)}(n)\right)$$

Como $y_i^{(3)}(n) = \varphi(v_j^{(3)}(n))$, é possível aplicarmos a regra da cadeia na derivada anterior para obtermos a equação (6).

$$s_i^{(3)} = -2\dot{y}_i^{(3)} \left(y_i^{(3)}(n) - R_i(n)\right)$$

$$s_i^{(3)} = -2\dot{\varphi}(v_i^{(3)}) \left(y_i^{(3)}(n) - R_i(n)\right) \tag{6}$$

Com o gradiente local da terceira camada de perceptrons, atualiza-se a matriz de pesos e *bias* conforme as equações (7) e (8). O termo η é a **taxa de aprendizado** (*learning rate*), que pode ser um valor fixo ou variável, portanto que seu valor esteja entre 0 e 1.

$$w_{ji}^{(3)}(k+1) = w_{ji}^{(3)}(k) - \eta\ s_j^{(3)} y_i^{(2)} \tag{7}$$

$$b_j^{(3)}(k+1) = b_j^{(3)}(k) - \eta\ s_j^{(3)} \tag{8}$$

Após definir os gradientes locais da terceira camada e atualizar sua respectiva matriz de pesos, é necessário propagar esta correção realizada para as demais camadas. Esta é a origem do nome do algoritmo. Seguindo a equação $s_i^{(m)} = \dot{\varphi}(v_i^{(m)})\ w_{ji}^{(m+1)}\ s_i^{(m+1)}$, obtemos os gradientes para as demais camadas nas equações (9) e (10).

$$s_i^{(2)} = \dot{\varphi}(v_i^{(2)})\ w_{ji}^{(3)}\ s_j^{(3)} \tag{9}$$

$$s_i^{(1)} = \dot{\varphi}(v_i^{(1)})\ w_{ji}^{(2)}\ s_j^{(2)} \tag{10}$$

Assim, também atualiza-se as matrizes peso e *bias* das demais camadas.

$$w_{ji}^{(2)}(k+1) = w_{ji}^{(2)}(k) - \eta\, s_j^{(2)} y_i^{(1)}$$
$$b_j^{(2)}(k+1) = b_j^{(2)}(k) - \eta\, s_j^{(2)}$$

$$w_{ji}^{(1)}(k+1) = w_{ji}^{(1)}(k) - \eta\, s_j^{(1)} y_i^{(0)}$$
$$b_j^{(1)}(k+1) = b_j^{(1)}(k) - \eta\, s_j^{(1)}$$

2.3 Learning rate

O processo de aprendizado repete-se em todas 6 mil imagens restantes, reservadas para o treinamento da rede neural. A cada análise de imagem, a rede melhora de acordo com a comparação entre os resultados obtidos e esperados. Neste trabalho, foi decidido que a taxa de aprendizado (do inglês, *learning rate*) seria constante ao decorrer do tempo. Em outros casos, pode-se escolher que sua *learning rate* varie de acordo com as iterações realizadas. O valor escolhido para a taxa de aprendizado é explicado mais adiante.

3 ESCOLHA DA *LEARNING RATE* E RESULTADOS

Definidas as quantidades de perceptrons em cada camada, (lembrando: 300, 100 e 10 perceptrons), realizou-se testes para a escolha da melhor learning rate considerando este caso. Na teoria de *learning rate* explicada em Hagan et al. (2014), o autor escolhe, como padrão, iniciar com um valor definido em $\eta = 0.1$. Usando este valor como referência, realizaram-se testes com suas variações próximas cuja saída da função fosse o erro na precisão das respostas. Para cada valor de *learning rate* foram criadas, do zero, cinco diferentes redes neurais, treinadas à partir da mesma base de 6 mil imagens de entrada, e calculou-se a média de erro de cada uma de acordo com a tabela 2. Para acelerar os cálculos dos erros, foram testadas apenas 785 imagens colhidas aleatoriamente da amostra de testes em cada caso da tabela.

Portanto, define-se $\eta = 0.06$ e com isto nossa rede neural desenvolve uma precisão de aproximadamente 87.64% no reconhecimento dos algarismos manuscritos. O desvio padrão dos resultados obtidos é de 2.96.

Quanto à classificação dos números, apresenta-se na figura 3 alguns exemplos de erros na identificação dos algarismos. A figura 4, por sua vez, apresenta exemplos onde a rede neural identificou com sucesso os algarismos da amostra de testes.

Tabela 2 – Erros obtidos para diferentes valores da *learning rate*.

η	1	2	3	4	5	Média
0,20	17,94 %	19,89 %	18,42 %	21,92 %	26,05 %	20,844 %
0,19	25,57 %	21,59 %	25,18 %	27,21 %	28,35 %	25,580 %
0,18	16,67 %	14,58 %	19,57 %	19,76 %	23,89 %	18,894 %
0,17	12,89 %	27,45 %	20,13 %	26,86 %	26,52 %	22,770 %
0,16	18,90 %	20,66 %	19,52 %	27,46 %	29,08 %	23,124 %
0,15	16,12 %	19,92 %	19,66 %	22,71 %	23,30 %	20,342 %
0,14	21,15 %	20,79 %	18,34 %	15,91 %	19,01 %	19,040 %
0,13	09,88 %	21,06 %	19,49 %	22,21 %	28,61 %	20,250 %
0,12	17,35 %	12,82 %	20,81 %	23,31 %	22,44 %	19,346 %
0,11	11,32 %	18,19 %	18,07 %	16,20 %	22,32 %	17,220 %
0,10	17,90 %	21,32 %	17,00 %	14,87 %	20,90 %	18,398 %
0,09	10,77 %	17,92 %	19,21 %	17,64 %	24,5 %	18,008 %
0,08	13,99 %	15,56 %	13,92 %	16,18 %	16,59 %	15,248 %
0,07	08,35 %	13,01 %	19,73 %	16,12 %	19,78 %	15,398 %
0,06	**05,22 %**	**12,51 %**	**13,16 %**	**18,51 %**	**12,40 %**	**12,360 %**
0,05	13,44 %	16,82 %	19,73 %	22,52 %	22,82 %	19,066 %
0,04	18,97 %	15,08 %	20,84 %	25,34 %	18,24 %	19,694 %
0,03	11,96 %	18,15 %	14,07 %	19,91 %	21,52 %	17,122 %
0,02	10,86 %	25,70 %	24,60 %	20,46 %	25,69 %	21,462 %

Figura 3 – Exemplos de algarismos lidos incorretamente pela rede neural. (a) real: 8, lido: 5; (b) real: 9, lido: 4.

(a) (b)

Figura 4 – Exemplos de algarismos lidos corretamente pela rede neural. (a)real e lido: 2; (b) real e lido: 5.

(a) (b)

4 CONCLUSÃO

Observando a variação nos erros da tabela 2, nota-se uma variação percentual muito grande num caso onde as redes deveriam gerar valores de erro mais estáveis.

Conforme ilustrado pelas figuras 3 e 4 apuradas pela rede neural, observamos que em alguns casos o erro acontece devido à falta de apuro na escrita; apesar de nós, humanos, sermos capazes de identificar os números em questão, algumas matrizes de computador podem ser confundidas facilmente. Acredita-se que uma maior amostragem de números “feios” podem treinar melhor a rede neural a fim de identificar estes casos.

REFERÊNCIAS

HAGAN, M. et al. *Neural Network Design*. 2. ed. [S.l.]: Martin Hagan, 2014.

HAYKIN, S. *Neural Networks and Learning Machines*. 3. ed. [S.l.]: Pearson, 2009.

NIELSEN, M. A. *Neural Networks and Deep Learning*. [S.l.]: Determination Press, 2015.