신경회로적인 전력조류 계산법에 대한 연구

김 재 주° 박 영 문
서울대학교 전기공학과

Load Flow Calculation by Neural Networks

Jae Joo Kim° Young-Moon Park
Seoul National University

Abstract

This paper presents an algorithm to reduce the time to solve Power Equations using a Neural Net. The Neural Net is trained with samples obtained through the conventional AC Load Flow. With these samples, the Neural Net is constructed and has the function of a linear interpolation network. Given arbitrary load level, this Neural Net generates voltage magnitudes and angles which are linear interpolation of real and reactive powers. Obtained voltage magnitudes and angles are substituted to Power Equations. Real and reactive powers are found. Thus, a new sample is generated. This new experience modifies weight matrix . Continuing to modify the weight matrix, the correct solution is achieved. Comparing this method with AC Load Flow, this method is faster. If we consider parallel processing, this method is far faster than conventional ones.

서론

전력조류 계산에는 해의 정확도와 함께 빠른 시간 내에 해를 구하는 것이 중요하다. 일반적으로 많이 사용하는 방법으로 AC Load Flow가 있는 데, 계통이 대형화되면 해의 정확도 면보다는 시간이 더 중요시된다. 즉, 해를 구하는데 걸리는 시간이 작게 걸리는 것이 필요하다. 그런데, 만일 병렬처리가 가능한 전력조류 계산법이 있다면 상당히 짧은 시간내에 해를 구할 수 있을 것이다. 그러나, 위의 방법은 병렬처리에는 부적합하다.

최근에, Neural Network가 상당히 각광을 받고 있는 데 그 주된 이유는 이 것이 병렬처리가 가능하다는 데 있다. 본 논문에서는 이 Neural Network를 사용하여 전력조류 계산을 하는 새로운 방법을 소개한다.

본론

1. Neural Network의 구조와 기능

(1) Neural Network의 일반적인 학습법

초기의 신경회로망에 대한 연구는, 학습이 가능하고 이 학습이 병렬적으로 진행되는 인간이나 동물의 신경활동을 수학적으로 모델링한 것으로부터 시작되었다. 학습은 주로 Training example이라고 불리는 것을 배움으로써 이루어진다. 신경회로망의 종류로는 Perceptron, Hopfield Net, Self organizing map 등이 있다. 그러나, 생물의 신경활동에 대한 기능을 모델링한 것은 좋으나, 아직은 이 인공적인 Network이 Training example을 완전하게 학습하도록 하는 알고리즘이 없는 실정에 머무르고 있다.

신경회로망에 대한 새로운 시각으로서, 1986년 Hecht-Nielsen은 비선형 방정식을 학습하게 할 목적으로 기존의 Self organizing map과 Stephen Grossberg의 Instar/Outstar Structure를 결합하여 Counterpropagation Network를 새로이 만들었다.

또한, 신경회로망을 '어떤 문제에 대한 병렬처리가 가능한 알고리즘이 있을 때 이것의 기능을 Network 형태로 표현한 것'이라는 관점으로 보고 새로운 접근을 시도하는 사람도 있다.

본 논문에서는 위의 새로운 시각에서 다음과 같은 세 가지 역할을 하는 신경회로를 구성한다.

첫째, Training examples을 정확하게 학습한다.

둘째, 학습하지 않은 입력에 대해서는 Linear Interpolation 역할을 한다.

세째, 비선형 전력조류 방정식을 참조하면서 학습하지 않은 입력에 대해서도 정확한 답을 출력한다.

(2) Neual Network의 구조

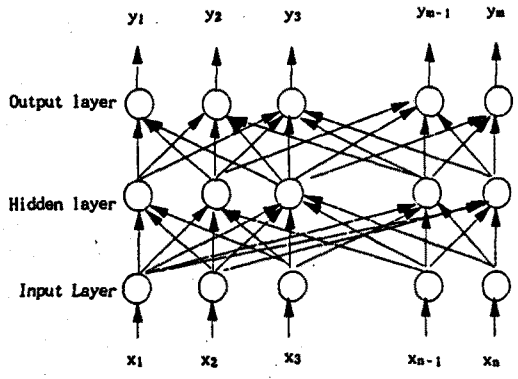
(그림 1)은 제안된 새로운 신경회로의 구조인데, 동그라미는 각각의 Neuron을 나타내고 각 Neuron간의 직선으로된 연결은 weight를 나타낸다. 이 Network는 3개의 Layer(층)로 구성되어 있는 데 맨 아래의 Layer는 입력층으로서 입력이 주어지고 맨 위의 Layer는 출력층으로서 출력이 나오게 된다. 이 Network의 수학적인 기능은 (식 1)-(식 3)에 해당한다.

$$z = W x \quad (1)$$

$$y = V z \quad (2)$$

$$y = L x \quad (3)$$

단, x : $n \times 1$ 입력 벡터
z : $n \times 1$ 내부 상태 벡터
y : $m \times 1$ 출력 벡터
W : $n \times n$ weight 행렬
$[w_{ij}]$ neuron x_j에서 neuron z_i로의 weight
V : $m \times n$ weight 행렬
$[v_{ij}]$ neuron z_j에서 neuron y_i로의 weight
$L = V W$: $m \times n$ 행렬



(그림 1) Linear Neural Net

(3) Training Example의 학습

신경회로망의 학습은 Training examples을 학습함으로써 이루어진다. Training example이란 입력에 대해 원하는 출력의 쌍을 나타낸다. 이제, 다음과 같이 n 개의 Training Example의 쌍 $(_1x,_1y),(_2x,_2y),\cdots\cdots,(_nx,_ny)$가 있다고 가정하자. 여기서, $(_ix,_iy)$는 i 번째 Training Example을 가리키고 x는 입력 벡터이고 y는 원하는 출력 벡터에 해당한다.

위의 Linear Neural Net으로 Training Example을 정확하게 학습할 수 있도록 weight matrix W,V를 정해보자.

먼저 n개의 Training Example에 대해 식 (1)-(3)을 만족해야 하므로

$$[_1y\ _2y\ \cdots\cdots\ _ny] = L\,[_1x\ _2x\ \cdots\cdots\ _nx] \qquad (4)$$
$$= V\,W\,[_1x\ _2x\ \cdots\cdots\ _nx] \text{ 이다.}$$

여기서, $V = [_1y\ _2y\ \cdots\cdots\ _ny]$ (5)

$W = [_1x\ _2x\ \cdots\cdots\ _nx]^{-1}$ (6)

라 정하면 (식 4)를 만족하게 된다. 단, $[_1x\ _2x\ \cdots\cdots\ _nx]$는 항상 역행렬이 존재함을 가정한다.

V matrix는 (식 5)에서 Training Example의 출력벡터로부터 간단히 구성할 수 있으나 W 행렬은 Training Example의 입력벡터들로 구성된 행렬의 역행렬이 된다. 이를 Gradient법의 일종인 Conjugate Gradient 방법으로 구한다.

(4) Conjugate Gradient법에 의한 역행렬

$$A = [_1x\ _2x\ \cdots\cdots\ _nx] \qquad (7)$$

라 하고 A의 역행렬이 반드시 존재한다고 가정하자.

이제 A W = I를 만족하는 W = A^{-1}를 구하면 된다. 여기서 weight 행렬 W의 j번째 column을 w^j라 하면

$$A\,w^j = I^j \qquad (\text{for } j = 1,2,\cdots,n) \qquad (8)$$

단,

$$I^j = \begin{bmatrix} 0 \\ 0 \\ \cdot \\ 1 \\ 0 \\ \cdot \\ 0 \end{bmatrix} \longleftarrow j\text{번째 element}$$

Error vector를

$$E^j = A\,w^j - I^j \qquad (\text{for } j = 1,2,\cdots,n) \qquad (9)$$

Performance Index를 F^j라 하고 다음과 같이 잡자.

$$F^j = 1/2\,(E^j)^T\,(E^j) \qquad (\text{for } j = 1,2,\cdots,n) \qquad (10)$$

weight 변화에 대한 Performance index의 Gradient vector

$$\nabla F^j = \partial F^j/\partial w^j$$
$$= A^T\,E^j \qquad (\text{for } j=1,2,\cdots,n) \qquad (11)$$

Conjugate Gradient법은 weight를 Gradient descent한 방향인 $-\nabla F^j$로 바꾸는 것이 아니라 새로운 Gradient vector h^j를 만들어서 이 방향으로 weight를 바꾸어 가는 것이다. 따라서,

$$h^j_k = -\nabla F^j_k + \beta^j_{k-1}\,h^j_{k-1} \qquad (12)$$

$$\beta^j_{k-1} = \frac{\langle \nabla F^j_k\ ,\ \nabla F^j_k \rangle}{\langle \nabla F^j_{k-1}\ ,\ \nabla F^j_{k-1} \rangle} \qquad (\text{for } j=1,2,\cdots,n) \qquad (13)$$

$\beta^j_{-1} = 0$ (단 , k는 iteration index를 나타낸다.)

$\langle\ \rangle$는 vector의 inner product operator

위에서 구한 Gradient h^j_k의 방향으로 weight를 바꾸어가자.

$$w^j_{k+1} = w^j_k + t^j_k\,h^j_k \qquad (\text{for } j=1,2,\cdots,n) \qquad (14)$$

t_k는 (식 10)을 최소로 하는 t_k를 사용한다.

따라서, $\partial F^j(w^j_{k+1})/\partial t_k = 0$ 인 t_k를 선택하면

$$\partial F^j(w^j_{k+1})/\partial t_k = (A\,h^j_k)^T\,(E^j_k + t_k\,h^j_k)$$
$$= 0$$

$$t^j_k = -\frac{\langle A\,h^j_k\ ,\ E^j_k \rangle}{\langle A\,h^j_k\ ,\ A\,h^j_k \rangle}$$
$$= -\frac{\langle \nabla F^j_k\ ,\ E^j_k \rangle}{\langle A\,h^j_k\ ,\ A\,h^j_k \rangle} \qquad (15)$$
$$(\text{for } j=1,2,\cdots,n)$$

(식 8) - (식 15)의 과정은 weight 벡터 w의 한 열씩 구하게 된다. 즉, 병렬처리가 가능하게 된다.

Conjugate Gradient법에 의해서 역행렬을 구하는 것은 Gauss Jordan법에 비해, (식 9)에서처럼 Error를 반영하면서 역행렬을 구하는 것이므로 이론적으로 Gauss Jordan법보다 더 정확한 역행렬을 구할 수 있다.

(5) 학습하지 않은 임의의 입력에 대한 Linear interpolation

신경회로망이 가지는 특징 중 하나는 학습하지 않은 입력에 대해서도 과거의 경험에 견주어 적당한 출력을 낸다는 것인데, 이전까지의 여러 신경회로망은 어떤 출력을 내는지 우리가 사전에 알 방법이 없다. 이에 비해서 본 논문에서의 방법은 학습하지 않은 입력에 대해 Linear interpolation 작용을 하도록 하였다.

(식 3)에서 입력 벡터 $x = [x_1\ x_2\ \cdots\ x_n]^T$의 $x_n = 1$ 이라고 가정하면 (식 16)을 얻을 수 있다.

$$y = \begin{bmatrix} l_{11} & l_{12} & \cdots & l_{1n-1} \\ l_{21} & l_{22} & \cdots & l_{2n-1} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ l_{n1} & l_{n2} & \cdots & l_{nn-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_{n-1} \end{bmatrix} + \begin{bmatrix} l_{1n} \\ l_{2n} \\ \cdot \\ \cdot \\ l_{2n} \end{bmatrix} \qquad (16)$$

임의의 입력이 주어질 경우 그것이 이미 학습한 것이라면 정확한 출력을 내게되고 학습하지 않은 것이라면 과거의 경험에 비추어 입력의 Linear interpolation에 해당하는 출력을 내게 된다.

(6) 새로운 Training Example의 획득에 의한 weight의 수정

본 논문에서의 신경회로망은 입력 벡터의 차원과 같은 수의 Training example이 필요한 데 새로운 Training example이 얻어지면 과거의 것을 대신하게 된다. 즉, 이 새로운 경험이 weight 행렬에 반영되게 된다.

이제, 새로운 x가 (식 5) - (식 6)의 j번째 열을 대신하게 된다고 가정하자. x를 V 행렬에 반영시키는 것은 j번째 열과 바꾸면 간단히 이루어지고 W 행렬에 반영시키는 것은 W의 한 열만 바꾸는 것이므로 Matrix Inverse Lemma를 사용하는 것이 경제적이다. A 행렬의 j번째 열 x^j을 x와 바꾼다면,

(식 7)과 (식 8)에서 $W = A^{-1}$ 이다.

Matrix Inverse Lemma는

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B\,(DA^{-1}B + C^{-1})^{-1}\,DA^{-1} \quad (17)$$

$$W_{new} = W - WB(DWB + C^{-1})^{-1}DW \quad (18)$$

여기서 $B = [x_1 - x_1^j \;\; x_2 - x_2^j \;\; \cdots \;\; x_n - x_n^j]^T$

$C = [\,1\,]$

$D = [\,0\,0 \cdots 1 \cdots 0\,0\,]$

위에서 D는 행 벡터로서 j번째 성분만이 1이고 나머지 부분은 모두 0 이다.

또한, 위의 알고리즘은 병렬처리가 가능하다.

2. 전력조류 방정식과 Training example의 획득

(1) 전력조류 방정식

전력조류 방정식은 (식 19)와 같다.

$$P_i + jQ_i = E_i \sum_{k=1}^{N} Y_{ik}^* E_k^* \quad (19)$$

$$= \sum_{k=1}^{N} |Y_{ik}||E_i||E_k| (\cos\theta_{ik} + j\sin\theta_{ik})$$

단, $\theta_{ik} = \delta_i - \delta_k - \delta_{ik}$

δ_i, δ_k 는 각기 i,k bus의 위상각

$\delta_{ik} = \arctan(B_{ik}/G_{ik})$

$Y_{ik} = \sqrt{G_{ik}^2 + B_{ik}^2}$

$|E_i|, |E_k|$ 는 각기 E_i, E_k의 크기

위의 방정식은 비선형 방정식으로 $|E|, \delta$가 주어질 경우 P,Q를 구하기는 대단히 쉬우나, Load flow 문제에서는 P,Q가 주어지고 $|E|, \delta$ 를 구하는 문제이므로 대단히 어렵다.

보통 AC Load Flow를 사용한다.

(2) Training examples의 획득

만약, n개의 bus와 m개의 탭이 있다고 가정할 경우, 여러 가지 부하 수준에서 AC Load Flow로써 해를 구하고, 이러한 많은 경험을 축적한 뒤에 다음과 같이 Training Example을 구성한다.

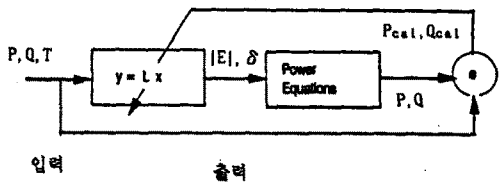
$$x = \begin{bmatrix} P_2 \\ \cdot \\ P_n \\ Q_1 \\ \cdot \\ Q_n \\ T_1 \\ \cdot \\ T_m \\ 1 \end{bmatrix}, \quad y = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \cdot \\ \cdot \\ \delta_n \\ |E_1| \\ |E_2| \\ \cdot \\ \cdot \\ |E_n| \end{bmatrix} \quad (20)$$

따라서, 위와 같은 경우에는 2n-1 + m 개의 Trainibg Example이 필요하다.

3. Training Example의 수정에 의한 해에의 수렴

(식 20)에서와 같이 입력과 출력 벡터를 구성하고 식 (5)와 (식 6)에 의해서 weight 행렬을 구성한다. 이제, 경험하지 않은 임의의 부하수준과 OPF를 반영한 결과에서 나온 탭 값으로 입력 벡터를 구성하여 (식 3)에 대입할 경우 Linear Interpolation에 의해서 전압과 위상각을 얻는다. 얻어진 이 전압과 위상각을 (식 19)의 Power Equation에 대입하면 P,Q를 얻는다. 이 P,Q, |E|, δ로 새로운 Training Example을 구성하게 된다. 이 Training Example을 2.(6)의 이론에 의해서 weight 행렬에 반영한다.

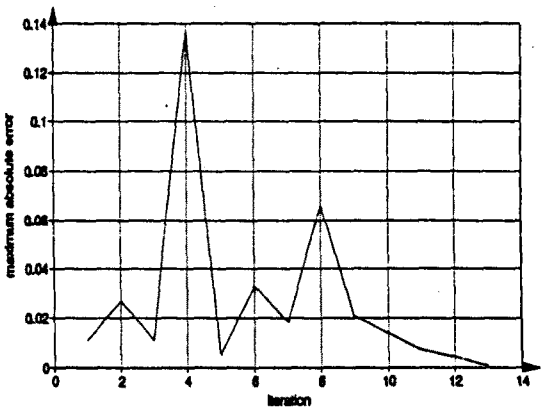
이러한 과정을 거치면서 weight를 계속 수정시켜 나가면 해에 수렴하게 된다. 아래의 (그림 2)는 전체적인 Block도이다.



〈그림 2〉 전체 구조도

4. 사례 연구

IEEE 30모선에서의 병렬처리를 고려하지 않은 시뮬레이션의 결과는 다음과 같다. (그림 3)은 임의의 부하수준과 탭 값에 대해서, 해에 해당하는 위상각과 전압값으로 수렴하는 과정을 나타낸 것으로, 가로축은 iteration의 수이며 세로축은 P,Q의 최대 오차를 나타낸 것이고 이 값이 10^{-3} 이하인 경우에 수렴한 것으로 본다.



5. 결론

사례연구의 결과 다음과 같은 결론을 얻었다.

첫째, 정확도에서는 AC Load Flow에서와 비슷한 정도의 오차인 10^{-3} 정도 이내의 해를 구할 수 있었다.

둘째, 수렴시까지 AC Load Flow는 7-8 iteration이 걸렸고 본 논문의 방법은 12 iteration이 걸렸다.

세째, n개의 bus를 가정할 경우 AC Load Flow에 의한 해법은 Jacobian 행렬이 약 2n x 2n 이고 차원이 늘어남에 따라 Gauss Jordan법에 의해서 역행렬을 구하므로 매 iteration 마다 약 $(2n)^3/3$의 계산이 필요하다. 본 논문의 방법은 매 iteration 마다 약 $3 \times (2n)^2$ 의 계산이 필요하다.

네째, 병렬처리를 고려할 경우에 입력 벡터와 같은 차원의 CPU를 둔다면 계산 시간은 매 iteration 당 3x(2n) 이 된다.

따라서, 본 논문의 방법은 시간면에서 기존의 방법보다 상당한 장점이 있다.

6. 참고문헌

[1] Hecht Nielsen, "Neurocomputing", Addison Wesley , 1989

[2] Yoh-Han Pao, "Adaptive Pattern Recognition and Neural Networks", Addison Wesley, 1989

[3] A.J.Wood, "Power Generation Operation and Control", 1983

[4] M.A.Pai, "Computer Techniques in Power System Analysis", 1979