특집원고

순환 신경망의 구조와 필기체 인식

서울대학교 ▌윤현구·정동석·정교민

1. 서 론

Recurrent Neural Network(RNN)은 인간의 신경세포의 동작과정을 모사한 neural network의 일종이다. 사용할 수 있는 데이터 양의 증가와 GPGPU의 등장과 같은 하드웨어의 성능 발전에 따라 RNN은 음성 인식 및 생성, 필기체 인식 및 생성, 자연어 처리 등 기존의 기계학습 알고리즘이 난항을 겪었던 문제들에서 뛰어난 성능을 보이며 주목받고 있다.

본 조사의 2장에서는 RNN의 특성에 대해 서술하였다. 2.1절에서 일반적인 모델과 표현식을 통해 RNN이 나타내는 시간성과 이로 인한 장점을 서술하였다. 2.2절에서는 RNN을 학습하는 가장 일반적인 알고리즘인 Back-Propagation Through Time(BPTT)의 개념을 서술하였다. 2.3절에서는 RNN을 학습할 때 나타나는 vanishing gradient problem에 대해 알아보고, 그 해결방법 중 한가지인 Long Short-Term Memory(LSTM)에 대해 2.4절에서 서술하였다.

3장에서는 LSTM-RNN을 이용한 실제 응용을 필기체 인식 사례를 통해 소개한다. 먼저 3.1절에서 필기체 인식 문제의 특징과 학습의 어려움을 알아본다. 3.2절에서는 이러한 어려움을 극복하기 위해서 사용한 CTC기법과 LSTM-RNN이 어떻게 적용되었는지 알아볼 것이다.

2. 회귀 신경망(RNN)

생물학적 관점에서 인간의 뇌는 <그림 1>에서 나타나는 것과 같이 뉴런들의 cyclic graph로 이루어져 있다[2]. 뇌는 다양한 행동, 신호처리, 자연어 처리 등과 같은 복잡한 작업들을 학습할 수 있는데, RNN은 여러 기계학습 모델 중 가장 뇌와 비슷한 형태의 모델이다. RNN은 Feed Forward Neural Network(FFNN)에 recurrent weight들을 추가하여 시간성을 모델 내부에 표현한 FFNN의 확대집합이다. 1991년 Siegelman과 Sontag는 RNN이 범용 튜링 기계를 시뮬레이션할 수 있다는 것을 증명하였다[3].

2.1 RNN의 일반적인 특성

기존의 FFNN 모델에서 output layer 이전 단계까지의 feed forward 과정은 nonlinear transformation이라 생각할 수 있다[1]. 모델의 각 노드들은 <그림 2>과 같이 nonlinear인 activation function으로 구성되어 있기 때문에 일반적인 FFNN 모델은 다음과 같은 식으로 표현될 수 있다.

$$x^{(l)} = f(W^{(l)}x^{(l-1)}) \tag{2.1}$$

$$\begin{aligned} out &= f^{out}(W^{out}f^{N-1}(in)) \\ &= f^{out}(W^{out}f(W^{(N-1)}f(W^{(N-2)}f(\cdots W^{(1)}in\cdots)))) \end{aligned} \tag{2.2}$$

(2.1)과 (2.2)에서 bias는 생략되어 있으며, l은 layer 번호, x는 각 layer의 출력신호벡터, out은 전체 모델의 출력, in은 전체 모델의 입력, W^n은 n번 째 layer로의

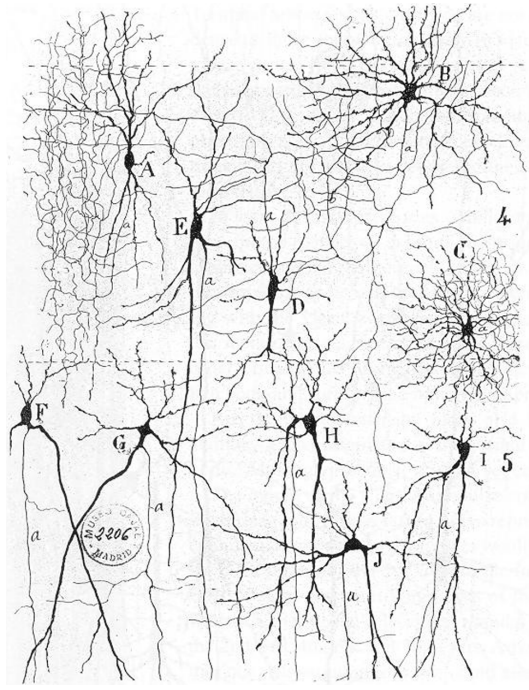

그림 1 인간 뇌의 뉴런 그래프모델[11]

weight 행렬, f는 각 노드들의 activation function이다. 위 (2.2)와 같이 FFNN에서는, 시간성이 있는 sequential data가 들어왔을 때, sequence와는 관계없이 오직 각 time step에서의 데이터로만 nonlinear transformation이 이루어지기 때문에 dynamical system의 특성을 보존하기 힘들다. 예를 들어 자연어 처리 시나리오에서, '기나긴 밤', '맛있는 밤' 2 개의 sequential data가 들어왔을 때, FFNN 모델은 앞서 있는 단어와 상관없이 2 개의 '밤' 이라는 단어를 똑같은 곳으로 nonlinear transformation하게 되며, 시간성을 고려하지 않은 FFNN 모델에서는 이와 같은 문제점을 해결할 수 없다.

RNN의 모델 구조는 <그림 3>과 같은 형태로 FFNN에 추가적인 recurrent weight(edge)들이 붙어있는 형태이다. Recurrent edge들은 해당 노드 자기 자신 또는 동일 layer의 노드들에게 신호를 전해주는 역할을 한다. 따라서 RNN 구조는 기존의 FFNN과는 다르게 cyclic graph로 나타내어지고, 구조 자체에 명백한 시간성을 띄게 된다. 다시 말하자면, RNN의 feed-forward propagation 단계에서 각 노드들의 state는 이전 time step에서의 노드들의 state에 직접적인 영향을 받게 된다. 이는 노드들의 state를 업데이트하는 다음 식에 분명하게 표현되어 있다.

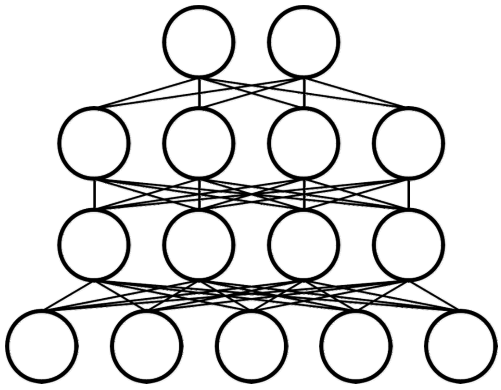

그림 2 일반적인 FFNN의 모델 예시

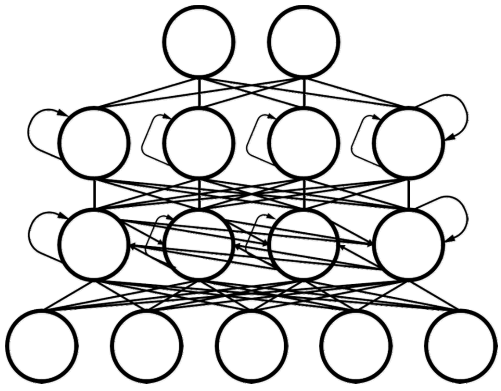

그림 3 RNN의 모델 예시

$$x_1^t = f(W_{ih_1} in + W_{h_1h_1} x_1^{t-1}) \quad (2.3)$$

$$x_n^t = f(W_{ih_n} in + W_{h_{n-1}h_n} x_{n-1}^t + W_{h_nh_n} x_n^{t-1}) \quad (2.4)$$

$$out^t = f^{out}(W_{ho} x^t) \quad (2.5)$$

bias는 생략되어 있으며, 윗첨자는 time step, 아랫첨자는 layer 번호이다. 위 (2.3)과 (2.4)에서는 FFNN에서는 없던 $W_{hh}x^{t-1}$ 항이 들어있는데, 이 항이 RNN의 시간성을 표현하고 있다. 따라서 RNN 모델에서는 '기나긴 밤'과 '맛있는 밤' 2개의 sequential data가 들어왔을 때, '기나긴' 뒤에 오는 '밤'과 '맛있는' 뒤에 오는 '밤'이 서로 다른 출력값을 갖게 된다.

또 다른 RNN의 장점 중 하나는 입력 데이터의 차원을 고정할 필요가 없다는 점이다. 학습이 끝난 RNN 모델 내부의 state 변화는 오직 W와 입력 데이터에 의해서 결정되는데 위 식들에서도 나타나 있듯이 W는 time step에 독립적이기 때문에 입력데이터의 차원이 고정되어 있지 않더라도 RNN은 동작한다. 2014년 Sutskever, Vinyals, V. Le는 영어-불어 간 문장-문장 통번역 문제에서 RNN을 이용해 입력 문장의 길이를 고정하지 않은 모델과 그 방법을 소개했다[4]. 저자들은 그들의 모델이 길이가 긴 문장도 문제없이 학습했다고 주장했다. 이와 같이 RNN은 문장-문장 간 번역, 필기체 인식 등과 같이 입력 데이터의 차원(길이)가 고정되지 않은 문제에서 효율적인 모델이다.

2.2 RNN의 학습 알고리즘

FFNN은 back-propagation 알고리즘을 통한 gradient descent 방법으로 모델 변수들을 학습한다[5]. 하지만 back-propagation 알고리즘은 모델에서 cycle이 존재하지 않다는 것을 가정하기 때문에 RNN에는 적용시킬 수 없다. RNN을 학습하는 방법들에 대해서는 지금도 많은 연구가 진행되고 있지만, 그 중 대표적인 방법은 Back-Propagation Through Time(BPTT)[6] 알고리즘이다.

BPTT는 back-propagation을 RNN에 적용시키기 위해 고안된 알고리즘이다. time step에 걸쳐서 RNN을 풀어내면 <그림 4>과 같은 구조를 가지게 된다. 이 때, hidden-hidden weight 행렬 W_{hh}과 input-hidden weight 행렬 W_{ih}, 그리고 hidden-output weight 행렬 W_{ho}는 모두 time step에 독립적이므로 <그림 4>에서 나타난 각각의 weight 행렬들은 모두 동일한 값을 지니게 된다. 풀어낸 RNN 구조는 cycle이 존재하지 않는 그래프, 즉 back-propagation 알고리즘을 적용시킬 수 있는 그래프이므로 기존의 FFNN과 같은 방식으로 모델을 학습할 수 있게 된다.

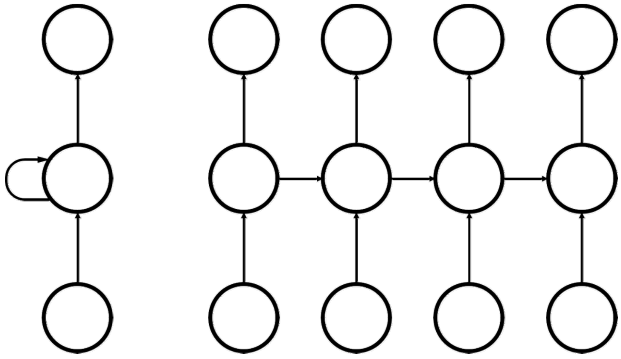

그림 4 간략히 나타낸 RNN(좌)와 time step에 따라 풀어낸 RNN(우)

BPTT 알고리즘을 적용시킬 때 총 T개의 time step 만큼을 고려한다면, 이는 T 개의 layer가 쌓여있는 구조를 학습시키는 것과 같다. 따라서 T가 커질수록 점점 더 깊은 구조를 학습시키는 것이기 때문에 RNN은 딥러닝의 일종이라고 볼 수 있다. 또한 딥러닝 구조를 학습할 때 발생하는 vanishing gradient 문제점들 역시 RNN을 학습할 때도 나타난다. 이런 문제점들을 해결하기 위해서, error propagation이 최대 τ 개의 time step까지만 이루어진다고 가정한 후 BPTT를 적용시키는 방법인 Truncated BPTT(TBPTT)가 있다.[7]

2.3 Vanishing Gradient Problem

Vanishing gradient problem은 layer가 깊게 쌓여있는 딥러닝 모델의 모델 변수들이 쉽게 학습되지 않는 주요한 이유이다. 많은 인공신경망 모델은 노드들의 활성화 함수로 sigmoid 함수($S(t)=\frac{1}{1+e^{-t}}$)를 사용하는데, 이는 <그림 5>에서 볼 수 있듯이 sigmoid 함수가 $(0,1)$ 사이의 값을 가지고 있어서 각 노드가 켜지거나 꺼질 확률값을 시뮬레이션 할 수 있는 성질을 가지고 있기 때문이다. 하지만 인공신경망 구조가 깊어질수록 깊은 layer의 대부분의 노드들은 1에 수렴하거나 0에 수렴하는 포화된 상태에 빠지게 된다. Sigmoid 함수 그래프에서 볼 수 있듯이 노드가 포화되어 있을 때, 즉 sigmoid의 입력이 매우 크거나 낮을 때, sigmoid 함수의 기울기는 0에 수렴한다. 이에 따라 back-propagation 알고리즘에서 핵심적인 역할을 하는 error propagation term $\delta(\frac{\partial err}{\partial s}$, err는 최종오차, s는 노드의 입력신호)가 0에 수렴하여 back-propagation 알고리즘이 원하는 대로 동작하지 않게 된다.

RNN에서 vanishing gradient problem은 [10]과 같은 관점에서 바라볼 수도 있다. [10]의 저자인 Alex Graves는

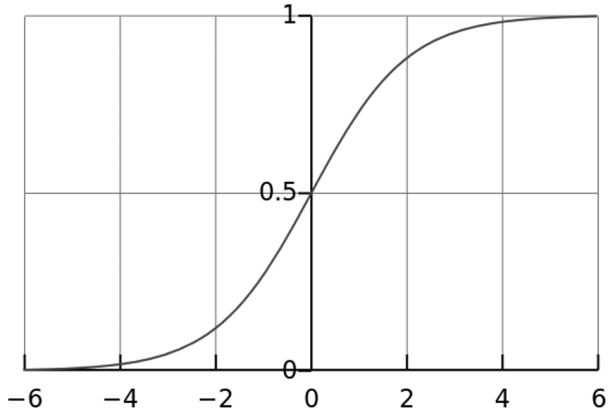

그림 5 Sigmoid 함수 개형[22]

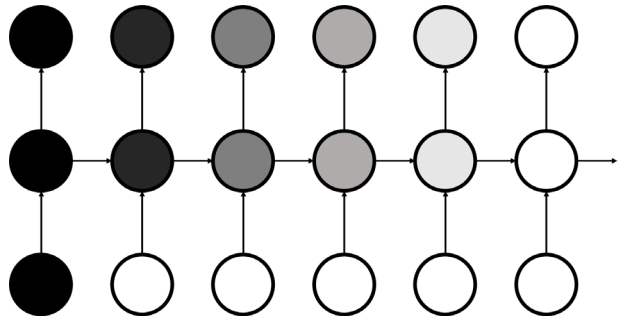

그림 6 Alex Graves가 해석한 RNN에서의 vanishing gradient problem[10]

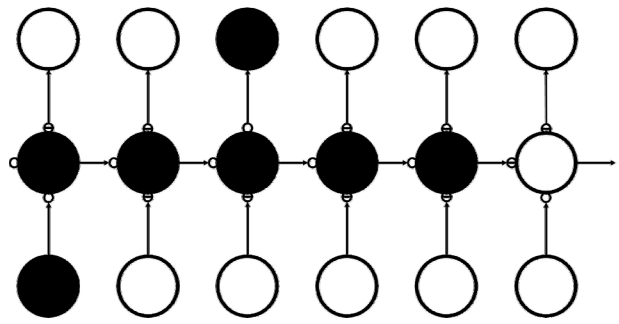

그림 7 Alex Graves가 해석한 LSTM의 효과[10]

RNN에서의 vanishing gradient problem은 time step에 따라 hidden layer의 정보가 점점 사라지는 것으로 해석했다. <그림 6>은 time step에 따라 풀어낸 RNN 도식도이다. 흰 노드는 정보 0을 갖고 있으며 검정 노드는 정보 1을 갖고 있는 것을 나타내고 있다. time step 1에서 1의 정보가 입력으로 들어오고 이 정보가 모두 hidden node로 전달되었으며 나머지 time step의 입력으로는 0이 들어왔다고 가정하자. W_{hh}와 W_{ih}는 [0,1] 사이의 값을 가지기 때문에 time step이 지날수록 hidden layer의 state 값은 점점 작아지다가 결국 time step 6에서는 state 값이 0에 수렴한다. 이는 time step 1에서 들어온 정보가 time step 6에서는 더 이상 남아있지 않는 것으

로 해석할 수 있으며, 따라서 제시된 모델에서는 최대 time step 개수가 5를 넘는($\tau > 5$) target 함수는 적절히 학습되지 않을 것이라는 것을 암시한다.

Convolutional Neural Network(CNN)과 같은 다른 딥러닝 모델에서는 vanishing gradient problem을 해결하기 위해서 일반적으로 Rectified Linear Unit(ReLU)를 활성화 함수로 사용한다[8]. ReLU[12]의 정의는

$$f(x) = \max(0, x) \fallingdotseq \ln(1 + e^x) \qquad (2.6)$$

으로, linear 함수를 사용하여 gradient가 0에 수렴하는 현상을 없애고 함수의 절반을 0과의 max 값을 취하여 non-linearity를 유지한다. RNN에서도 활성화함수로 ReLU를 사용하였을 때 성능 향상이 있었다는 연구가 2013년에 Bengio et al.에 의해 발표되었다[9].

2.4 Long Short-Term Memory

일반적으로 RNN에서는 vanishing gradient problem을 예방하기 위한 활성화함수로 Long Short-Term Memory (LSTM)[13]을 사용한다. LSTM은 각 노드로 흘러 들어가는 정보의 양을 gate들을 이용해 조절하여 정보가 희석되는 현상을 늦춰준다. Alex Graves는 [10]에서 <그림 7>을 제시하며 LSTM의 효과를 다음과 같이 해석했다. <그림 7>은 hidden node에 gate들이 추가된 형태의 RNN이다. 이 gate들은 gate로 들어오는 정보와 나가는 정보의 양을 결정하는데, gate가 열려있는 'O' 상태는 모든 정보가 들어오거나 나가고, gate가 닫혀있는 '⊖' 상태는 어떤 정보도 들어오거나 나갈 수 없다고 가정한다. 따라서 <그림 7>의 time step 1에서 input에 정보 1이 들어오면, hidden 노드 역시 열려있는 gate를 통해서 1의 정보를 가지게 된다. 하지만 hidden-output으로 나가는 gate가 닫혀있기 때문에 output 노드는 0의 정보를 갖게 된다. 그에 비해 time step 2에서는 input으로 0의 정보가 들어와도 input-hidden gate가 닫혀있기 때문에 hidden 노드에 아무런 영향을 주지 못하고 hidden 노드는 time step 1의 hidden 노드로부터 받은 1의 정보를 갖고 있게 된다. 따라서 앞선 <그림 6>의 RNN과는 달리 LSTM-RNN은 이와 같은 방식으로 gate가 열리고 닫히는 정도, 즉 hidden 노드의 정보가 희석되는 정도를 조절할 수 있기 때문에 vanishing gradient 현상을 어느 정도 예방할 수 있다.

LSTM 노드(혹은 block) 하나의 구조는 다음 <그림 8>과 같다[10, 14]. LSTM 노드의 중앙에는 cell이 존재하여 LSTM 노드의 정보(state)를 저장하는 역할을 한다. a는 입력, b는 출력을 나타내며, cell, input 게이트, output 게이트, forget 게이트는 각각 아랫 첨자 c, ι, ω, ϕ로 표시한다. 또한, cell에 들어있는 정보, 즉 cell의 출력값은 s_c로 표시한다. 윗첨자는 time step을 나타낸다. Input 게이트는 a_c^t가 cell state에 얼마만큼의 영향을 미칠지를 결정한다. Forget 게이트는 cell state가 자신의 정보를 얼마만큼 기억할지를 결정한다. Output 게이트는 cell state를 얼마만큼 출력할 것인지를 결정한다. 이를 식으로 표현하면 다음과 같다.

*Input gate*의 입력 :

$$a_\iota^t = W_{i\iota} in^t + W_{h\iota} b_h^{t-1} + W_{c\iota} s_c^{t-1} \qquad (2.7)$$

*Forget gate*의 입력 :

$$a_\phi^t = W_{i\phi} in^t + W_{h\phi} b_h^{t-1} + W_{c\phi} s_c^{t-1} \qquad (2.8)$$

*Output gate*의 입력 :

$$a_\omega^t = W_{i\omega} in^t + W_{h\omega} b_h^{t-1} + W_{c\omega} s_c^t \qquad (2.9)$$

*Input gate*의 출력 : $b_\iota^t = f_\iota(a_\iota^t)$ (2.10)

*Forget gate*의 출력 : $b_\phi^t = f_\phi(a_\phi^t)$ (2.11)

*Output gate*의 출력 : $b_\omega^t = f_\omega(a_\omega^t)$ (2.12)

*LSTM block*의 입력 : $a_h^t = W_{ic} in^t + W_{hc} b_h^{t-1}$ (2.13)

*Cell*의 출력 : $s_c^t = b_\phi^t s_c^{t-1} + b_\iota^t f_c(a_h^t)$ (2.14)

*LSTM block*의 출력 : $b_h^t = b_\omega^t f_h(s_c^t)$ (2.15)

여기서 $W_{c\iota}$, $W_{c\phi}$, $W_{c\omega}$는 Gers et al이 제시한 peephole weight[15]로서, cell state가 각각의 게이트들에게 미치는 영향을 나타내며, LSTM 노드가 좀 더 복잡한 모델을 학습할 수 있게 해준다. <그림 8>에서는 점선으로 표시되어 있다.

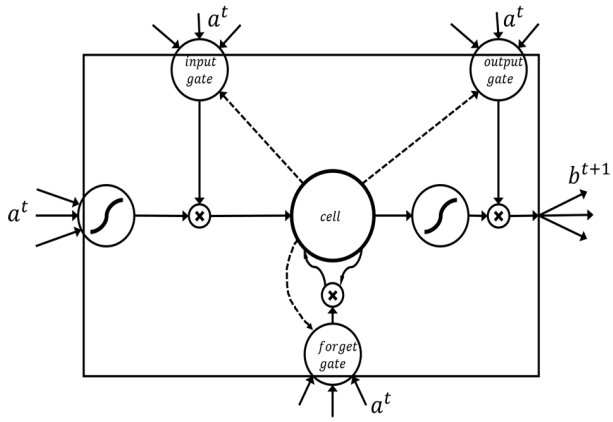

그림 8 LSTM block의 내부 구조[14]

3. RNN을 이용한 필기체 인식[1)]

순차적인 입력을 학습하는 문제 중에서 대표적인 것으로 필기체 인식이 있다. LSTM-RNN구조를 사용하여 필기체 인식의 성능을 획기적으로 향상시킨 Alex Graves의 연구사례를 바탕으로 RNN이 순차적 입력을 가지는 데이터를 어떻게 처리하는지 알아본다.[16]

3.1 필기체 인식의 특징

기본적으로 필기체 인식과 같은 시간성이 있는 sequential data의 경우 입력 데이터의 크기가 정해져 있지 않기 때문에 기존의 고정된 크기의 입력을 가지고 학습하는 알고리즘으로는 학습이 어렵다. 특히 음성인식, 필기체 인식 등과 같이 연속적인 시간성을 가지는 데이터에서는 Sayre의 딜레마라고 알려진 문제점이 나타난다. 일반적인 학습의 경우 적절하게 분할되어 라벨을 가지는 입력 값에 의한 출력을 바탕으로 학습을 실시한다. 하지만 음성 인식, 필기체 데이터와 같이 연속된 시간성을 가진 데이터의 경우 학습을 위해선 이를 적절한 단위로 '분할'하여야 하는데 '분할'하기 위해서는 해당 부분이 어떤 라벨을 나타낼지 '인식'되어야 한다.[16]

다음 <그림 9>의 필기체 데이터는 m과 n으로 이루어진 문자열들이다. 이 필기체 문자열을 일반적인 문자열 인식 알고리즘으로 인식하기 위해서는 한 문자가 표현하는 구간을 정하고, 그에 따라 구간별로 해당 구간이 어떤 문자를 판별하는지 확인하여야 한다. 하지만 이 문제의 경우 해당 문자가 'm'인지 'n'인지 알기 전에는 그 구간을 자르는 것이 어려운 문제임을 알 수 있다. 즉 해당 문자를 'm'인지 'n'인지 구분하기 전에는 데이터를 분할하는 것이 불가능하고, 데이터를 분할하기 전에는 각각의 데이터가 어떤 문자인지 구분하는 것이 불가능하다는 모순이 생긴다.

그림 9 필기체의 인식과 분할(mn, nnm, nmn, mnn)[23]

3.2 LSTM-RNN과 필기체 인식

LSTM-RNN의 구조와 학습방법에 대해서는 언급을 하였으므로 LSTM-RNN이 어떻게 필기체 인식을 학습하는지 알기 위해서는 주어진 네트워크 구조에 어떤 입력을 넣어주고, 목적함수를 어떻게 정하는지만 알면 LSTM-RNN이 필기체 인식을 하는 과정을 이해할 수 있다.

필기체 데이터를 처리하기 위해선 글씨의 비뚤어짐(skew), 기울어짐(slant), 높이, 넓이 등 필기 스타일이 달라질 수 있기 때문에 데이터를 정규화(normalization)하는 과정을 거친다. <그림 10>은 각각 데이터 정규화를 하기 전과 후를 나타낸다.

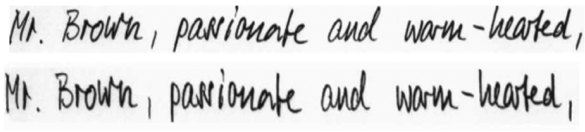

그림 10 데이터 정규화 전(위), 데이터 정규화 후(아래)[17]

그 후 sliding window방식으로 특징 벡터를 추출한다. 한 픽셀씩 윈도우를 이동해가며 다음 9가지 기하학적 특징 값을 계산하여 해당 RNN의 해당 time step의 네트워크의 입력으로 넣는다.[16]

- 픽셀들의 평균 흑백 값(mean gray value)
- 픽셀들의 무개중심
- 무개중심의 가로방향 2차 모멘트
- 가장 위쪽의 검은 픽셀 위치
- 가장 아래쪽의 검은 픽셀 위치
- 이러한 지점들의 변화율

앞서 언급한 필기체 인식문제에서 나타나는 Sayre의 딜레마를 해결하기 위해서 Connectionist Temporal Classification(CTC)라고 불리는 기법이 사용되었다[18]. 전통적인 RNN의 목적 함수는 미리 잘 분리된 입력에 대하여 목표로 하는 출력을 내도록 하였다. 따라서 이런 문제 같은 경우에는 미리 각각의 문자 단위로 입력을 잘라서 넣어줘야 제대로 학습이 되는 것이다. CTC기법은 시퀀스 라벨링 작업을 위하여 다자인된 RNN의 output layer이다. CTC기법을 간단히 설명하자면 각각의 time step의 output layer의 출력을 각각의 '라벨' 혹은 '공란'이 될 '확률'로 계산하고, 연속적으로 같은 라벨이 나오는 경우를 같은 문자로 엮은 다음 학습하고자 하는 정답 시퀀스가 나올 확률을 최대화 하도록 목적함수를 정하는 것이다. 이러한 기법을 사용한 결과 <표 1>과 같이 기존의 HMM 기법보다 월등히 뛰어난 성능을 보이는 것으로 나타났다.[16]

표 1 HMM기법과 RNN 기법의 필기체 단어인식 정확도

기법	단어 예측 정확도
HMM	64.5%
RNN	74.1±0.8%

1) 필기체 인식은 디지털 입력기로 입력되는 실시간 데이터를 학습하는 on-line 필기체 인식과 이미 쓰인 이미지를 학습하는 off-line 필기체 인식이 있다. 본 조사에서는 논의의 편의를 위해 off-line의 경우만 다루었다.

4. 결 론

지금까지 RNN과 LSTM의 구조와 학습 알고리즘을 살펴보고, 필기체 인식의 연구사례를 통해 시간성을 가지는 데이터를 어떻게 학습시키는지 알아보았다. 요약하자면, RNN은 기존의 neural network 모델에서 시간성을 분명하게 추가한 모델이며, 이에 따라 생기는 recurrent weight들 때문에 Back-Propagation 알고리즘을 바로 적용할 수 없지만, 모델을 time-step에 따라 풀어내는 트릭을 사용한 Back-Propagation Through Time 알고리즘을 적용하여 학습시킨다. Time-step에 따라 풀어낸 모델은 필연적으로 심층 학습 구조를 가지게 되며 다른 많은 심층 학습 구조와 같이 vanishing gradient problem이 발생하기 때문에 이를 막기 위해 LSTM을 노드의 활성화 함수로 사용한다. 필기체 인식을 통해 알아본 활용 예에서는 Sayre의 딜레마라고 알려진 연속적인 시간성을 가지는 데이터를 합성할 때 나타나는 어려움과 그 해결방법에 대해서 살펴보았다.

이러한 RNN 관련 연구는 음성 인식, 필기체 인식, 자연어 처리 분야에서 최고 성능을 기록하였으며, 최근에는 새로운 문제(프로그램 출력 학습)[19]등의 새로운 분야에 RNN을 적용시키거나, 새로운 활성화 함수 모델(GRU)[20]와 같이 노드의 구조를 변경하여 새로운 효과를 기대하거나, 새로운 네트워크 모델(GF-RNN)[21]처럼 모델의 연결 구조를 변경하여 long-term time dependency를 학습하는 등의 연구가 이루어지고 있다.

특히 신경망 구조를 이용한 기계학습 기법은 계산을 통한 '인공 지능'의 구현이라는 컴퓨터공학 분야의 오랜 꿈과도 맞닿아 있다. 앞으로도 RNN을 이용한 심층학습 기법들은 활발한 연구가 이루어 질 것으로 기대된다.

참고문헌

[1] Bishop, Christopher M. Pattern recognition and machine learning. springer, 2006.

[2] Stam, Cornelis J., and Jaap C. Reijneveld. "Graph theoretical analysis of complex networks in the brain." Nonlinear biomedical physics 1.1 (2007): 3.

[3] Siegelmann, Hava T., and Eduardo D. Sontag. "Turing computability with neural nets." Applied Mathematics Letters 4.6 (1991): 77-80.

[4] Sutskever, Ilya, Oriol Vinyals, and Quoc VV Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.

[5] Hagan, Martin T., and Mohammad B. Menhaj. "Training feedforward networks with the Marquardt algorithm." Neural Networks, IEEE Transactions on 5.6 (1994): 989-993.

[6] Paul J Werbos. Backpropagation through time: what it does and how to do it. Proceedings of the IEEE, 78(10):1550{1560, 1990.

[7] Williams, Ronald J., and David Zipser. "A learning algorithm for continually running fully recurrent neural networks." Neural computation 1.2 (1989): 270-280.

[8] Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." Proceedings of the 27th International Conference on Machine Learning (ICML-10). 2010.

[9] Bengio, Yoshua, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. "Advances in optimizing recurrent networks." Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.

[10] Graves, Alex. Supervised sequence labelling with recurrent neural networks. Vol. 385. Heidelberg: Springer, 2012.

[11] Cajal, Santiago Ry, Pedro Pasik, and Tauba Pasik. Texture of the Nervous System of Man and the Vertebrates: I. Vol. 1. Springer Science & Business Media, 1999.

[12] Wikipedia. Rectifier (neural network) - Wikipedia, the free encyclopedia, 2015. [Online; accessed 22-July-2015]

[13] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[14] Graves, Alan, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks." Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.

[15] Gers, Felix A., Nicol N. Schraudolph, and Jürgen Schmidhuber. "Learning precise timing with LSTM recurrent networks." The Journal of Machine Learning Research 3 (2003): 115-143.

[16] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber. "A novel connectionist system for unconstrained handwriting recognition", IEEE Transactions on Pattern Analysis and Machine

Intelligence, Volume 31 Issue 5, May 2009 pp. 855-868.
[17] Kenneth M. Sayre, “Machine Recognition of Handwritten Words: A Project Report,” Pattern Recognition, Pergamon Press, Vol. 5, 1973, pp. 213-228.
[18] A. Graves, S. Fernández, F. Gomez, J. Schmidhuber. “Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks”, ICML 2006, Pittsburgh, USA, pp. 369-376.
[19] Zaremba, Wojciech, and Ilya Sutskever. "Learning to execute." arXiv preprint arXiv:1410.4615 (2014).
[20] Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." arXiv preprint arXiv:1412.3555 (2014).
[21] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, Yoshua Bengio “Gated Feedback Recurrent Neural Networks”, ICML 2015, Lille, France, pp. 2067 – 2075.
[22] https://en.wikipedia.org/wiki/Sigmoid_function
[23] http://www.cs.toronto.edu/~graves/handwriting.html

약 력

윤 현 구

2015 서울대학교 전기·정보공학부(학사)
2015～현재 서울대학교 전기·정보공학부(석·박사 통합과정)
관심분야: 머신 러닝, 심층학습 구조
Email: youaredead@snu.ac.kr

정 동 석

2014 서울대학교 전기·정보공학부(학사)
2014～현재 서울대학교 전기·정보공학부(석·박사 통합과정)
관심분야: 머신러닝, 심층학습 구조, 언어 모델
Email: dongseok.j@snu.ac.kr

정 교 민

2003 서울대학교 수학과(학사)
2009 MIT 수학과(박사)
2009～2013 KAIST 전산학과 교수
2013～현재 서울대학교 전기정보공학부 교수
관심분야: 머신 러닝, 심층학습 구조, Social Network 정보 확산 분석 및 예측
Email: kjung@snu.ac.kr