9 장: 최신 딥러닝 구조

9.1 뉴럴 튜링 머신
9.2 메모리넷
9.3 생성 대립넷
9.4 인코더-디코더 모델
9.5 딥 잔차넷

2010 년 이후로 딥러닝이 인공지능의 여러 난제를 해결할 새로운 돌파구가 될 수 있다는 기대는 다양한 분야에서 딥러닝을 활용하는 촉매제와 같은 역할을 하였다. 토론토대, 뉴욕대, 스탠포드대, 몬트리올대 등, 딥러닝 연구의 허브와도 같은 역할을 수행하였던 아카데미뿐만 아니라 구글, 페이스북, 바이두와 같은 IT 기업들의 연구소에서도 다양하고 참신한 연구가 이어졌다. 9.1 절에서는 앨런 튜링의 튜링 머신의 딥러닝 버전인 뉴럴 튜링 머신을 살펴본다. 9.2 절에서는 인간의 기억 기제를 모사하여 기억 부분과 인출 부분을 다르게 학습하도록 구현한 메모리넷을를 살펴본다. 9.3 절에서는 최근 생성 모델 연구에서 가장 혁신적인 방법으로 알려진 생성 대립넷을 살펴본다. 9.4 절에서는 기계 번역 등에서 활용하고 있는 인코더-디코더 모델을 살펴본다. 9.5 절에서는 여러 시각 과제 챌린지에서 1 위를 차지하며 2015 년의 주요 넷 모델로 자리잡은 딥 잔차넷을 살펴본다.

9.1 뉴럴 튜링 머신

일반적으로 정의된 컴퓨터 프로그램에는 세 가지 중요한 구성요소가 있다. 첫째는 일반적인 연산을 하는 산술 연산이고, 둘째는 조건부 분기를 담당하는 논리 흐름 제어, 그리고 마지막으로 데이터를 저장하는 외부 저장장치가 그것이다. 하지만 현대의 기계 학습론에서는 산술연산에 중시하여 입력 정보를 어떤 추상화된 표현에 사상하는 방법을 주로 사용하며, 조건부 분기 및 외부 저장장치는 중요하게 고려하지 않고 있다.

순환신경망은 입력 패턴들의 시간적인 순서를 고려한다는 점에서 여타 다른 기계학습 알고리듬과는 다른 면모를 보인다. 또한 기본적인

순환신경망은 이론적으로는 모든 시간 단위에 대해서 누적연산이 수행되므로 튜링-완전기계(Truing-complete machine)의 조건을 가지고 있으나(Graves, 2014), 현실적으로는 그러한 조건을 충족시키기에는 부족한 면이 있다. 이런 문제를 해결하기 위해서는 튜링 머신에서 제안한 것처럼 무한의 메모리 테이프를 가지는 것이 필요하다. 이를 근사적으로 해결하는 방법으로 매우 크고 직접적으로 참조 가능한 외부 메모리를 가지면서도 인공 신경망구조의 제어기를 활용하는 시스템을 만들었으며, 이 시스템을 뉴럴 튜링 머신이라고 부른다(Siegelmann, 1992). 이러한 뉴럴 튜링 머신을 이용하여 복사, 정렬, 연관기억 연상 등의 간단한 알고리듬을 구현해 낼 수 있다. 또한 이 모델로 구성된 시스템의 경우 구조 자체는 기존의 튜링 머신이나 폰 노이만 구조와 비슷하지만 미분이 가능하다는 중요한 차이점이 있는데 이는 곧, 뉴럴 튜링 머신이 효율적인 학습 방법으로 널리 알려진 경사 하강기법을 통해 학습 될 수 있다는 것을 의미한다.

9.1.1 튜링 머신

1928 년 독일의 수학자 힐버트는 다음과 같은 문제를 제기했다.

'원칙적으로 수학의 모든 문제를 순서대로 해결할 수 있는 일반적인 기계적 절차가 있는가?'

튜링 머신은 튜링이 이러한 질문에 답하기 위해 구상해낸 일종의 추상적인 계산기계이다. 이러한 튜링 머신은 정해진 절차에 따라 특정 계산이나 논리 연산을 수행하는 장치로 충분한 기억장소와 적절한 알고리듬만 주어진다면 어떠한 계산이라도 가능함을 보여줌으로써 현대 컴퓨터의 원형을 제시했다. 튜링 머신은 1936 년 *"On Computable Numbers, with an Application to the Entscheidungs-problem"* 이라는 논문에 발표되었으며 이 논문을 요약하면 다음과 같다(wikipedia-Turing_machine).

1. 셀이라고 불리는 정사각형 칸으로 나누어진 연속된 테이프가 있다고 가정하자. 각각의 셀은 유한수의 기호들을 포함할 수 있다. 또한 테이프는 왼쪽 또는 오른쪽으로 필요에 따라 항상 확장될 수 있다. 즉 테이프의 길이는 무한대라고 가정한다. 아무것도 기록되지 않은 셀은 빈칸임을 의미하는 특수한 기호가 쓰여 있다.

2. 헤드는 테이프에 쓰여진 기호들을 읽거나 테이프에 쓸 수 있으며 한번에 한 칸의 셀로 이동할 수 있다.
3. 튜링 머신의 상태를 저장하고 있는 상태 저장소가 있다. 이 상태 저장소의 상태들을 인간이 계산을 수행할 때 인간 마음의 상태라고 비유한다.
4. 튜링 머신이 특정 작업을 완료하기 위해 필요한 유한개의 명령어 테이블이 있다.

뉴럴 튜링 머신은 이러한 튜링 머신의 개념에서 영감을 받아 만들어 진 실제적인 시스템이다. 또한 뉴럴 튜링 머신은 이론적으로 튜링-완전기계의 조건을 만족하는 재귀신경망을 제어기로 사용하며, 일반적으로 인공 신경망 모델에서 다루고 있지 않은 외부 메모리의 역할을 결합한 모델로써 미분 가능한 튜링 머신으로 볼 수 있다.

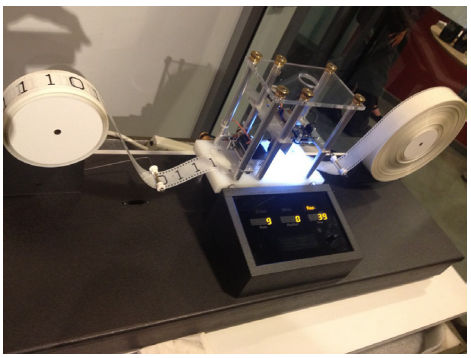


그림 9.1 튜링 머신

9.1.2 뉴럴 튜링 머신

뉴럴 튜링 머신은 튜링기계와 비슷하게 두 가지 주된 구성요소인 제어기(controller)와 기억장치(external memory)가 있다. 그림 9.2 에서는 이러한 뉴럴 튜링 머신의 구조를 간략하게 보여준다. 제어기는 일반적으로 인공 신경망 기반의 모델을 사용하며 7 장에서 다룬 장단기 메모리 모델을 주로 사용한다. 따라서 제어기의 입력과 출력은 모두 실수 벡터 형태로 표현한다.

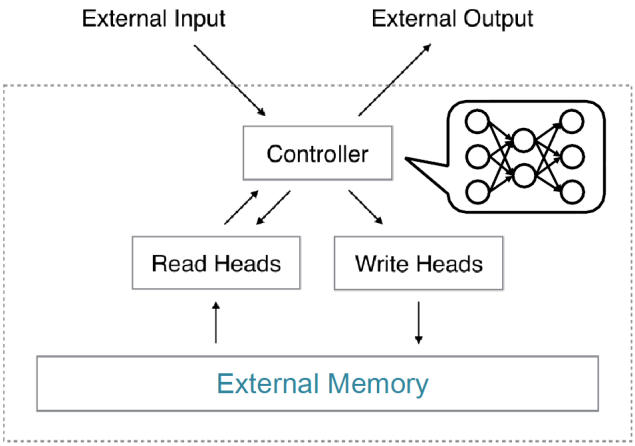


그림 9.2 뉴럴 튜링 머신 구조

(그림 출처: http://www.slideshare.net/yuzurukato/neural-turing-machines-43179669)

앞서 설명한 바와 같이, 뉴럴 튜링 머신은 일반적인 인공신경망 모델과는 다르게 읽기, 쓰기 연산을 사용하여 외부 메모리와 직접적으로 상호작용한다. 또한 매개변수화 된 제어기의 출력연산들을 튜링 기계의 헤드에 비유할 수도 있다.

튜링 기계와 구분되는 뉴럴 튜링 머신의 큰 특징 중 하나는 모든 구성 요소가 미분이 가능하다는 점이고 이는 곧, 경사 하강 법을 통해 학습 할 수 있다는 것이다. 이는 읽기와 쓰기 연산에 선택적 집중 매커니즘을 도입한 결과로 달성할 수 있었으며, 전체 메모리의 일부분만을 특정 지어 접근할 수 있다는 장점이 있다. 이러한 선택적 집중 매커니즘은 읽기와 쓰기 연산에 가중치를 도입한 것이며 이는 곧, 메모리에 접근하는 단계에서 한 부분에 강하게 집중하고 남은 여러 부분에는 약하게 집중하는 효과를 줄 수 있다는 말이 된다.

9.1.3 뉴럴 튜링 머신의 읽기 및 쓰기

뉴럴 튜링 머신의 읽기와 쓰기 연산에 대해 개략적으로 나타내면 그림 9.3 과 같다.

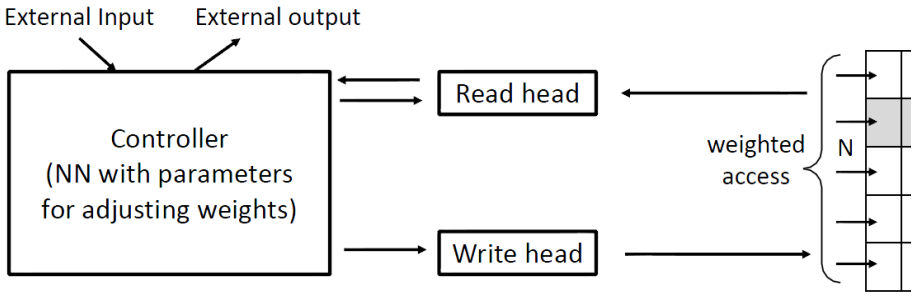


그림 9.3 읽기와 쓰기연산의 개략

(그림 출처: http://www.slideshare.net/yuzurukato/neural-turing-machines-43179669)

M_t 는 시각 t 에서의 $M \times N$ 크기의 메모리 행렬을 의미하며, N 은 메모리 슬롯의 크기를 나타내고, M 은 각각의 슬롯에서 데이터 벡터의 크기를 의미한다. 또한, w_t 를 시각 t 때 읽기 헤드로부터 나온 모든 N 위치에 대한 가중치라고 하자. 이때, 모든 가중치들을 정규화 하여 N 번째 위치에 대한 가중치 $w_t(i)$ 는 다음 조건을 따른다.

$$\sum_i w_t(i) = 1, \qquad 0 \le w_t(i) \le 1, \forall i.$$

읽기 헤드로부터 나오는 M 길이의 읽기 벡터 r_t 는 메모리의 행 벡터 $M_t(i)$ 들의 컨벡스 조합으로 정의되며 다음과 같다.

$$r_t \leftarrow \sum_i w_t(i) M_t(i),$$

이는 가중치와 메모리에 대해 명확하게 미분이 가능한 형태이다.

뉴럴 튜링 머신에서 메모리에 쓰는 작업의 경우 장단기 메모리모델의 입력 및 망각 게이트의 개념을 이용하여 삭제와 추가 연산으로 분할해서 생각해 볼 수 있다. 쓰기 헤드로부터 시각 t 에서 나오는 가중치 w_t 가 주어졌을 때, 모든 요소가 0~1 사이의 값을 가지는 M 길이의 삭제 벡터 e_t 와 이전 시간 단계의 메모리 벡터 $M_{t-1}(i)$ 를 이용하여 현재의 메모리를 수정 할 수 있다. 우선 다음을 계산한다.

$$\tilde{M}_t(i) \leftarrow M_{t-1}(i)[1 - w_t(i)e_t],$$

위 식에서 1 은 모든 요소 값이 1 인 행 벡터를 의미하고, 메모리에 적용되는 곱셈은 요소 단위의 스칼라 곱셈을 의미한다. 따라서, 각 메모리의 요소들은 그 위치의 가중치와 삭제 벡터모두 1 인경우 0 으로 변환된다 (반면, 가중치 및 삭제벡터 모두 0 인 경우 변화 없음). 삭제 작업 이후 쓰기 작업을 수행하는데 이는, M 길이의 추가 벡터 a_t 를 이용하여 $M_t(i)$ 를 다음과 같이 계산할 수 있다.

$$M_t(i) \leftarrow \tilde{M}_t(i) + w_t(i)a_t$$

위에서 정의한 삭제 및 추가연산을 모든 쓰기 헤드에 대해 수행하면 최종적으로 시각 t 에서의 메모리 M_t 를 계산할 수 있다. 또한 삭제 및 추가연산 모두 미분 가능한 형태이므로 쓰기연산 역시 미분 가능하다.

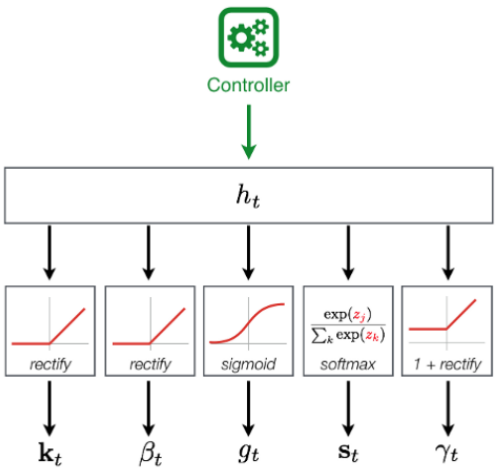


그림 9.4 가중치 갱신을 위한 변수들

(그림 출처: https://medium.com/snips-ai/ntm-lasagne-a-library-for-neural-turing-machines-in-lasagne-2cdce6837315#.r921877sp)

9.1.4 뉴럴 튜링 머신의 주소지정 매커니즘

지금까지 읽기 및 쓰기 연산에 대해 알아보았다. 하지만 뉴럴 튜링 머신의 동작을 이해하기 위해 한가지 더 알아야 할 것은 가중치 벡터 w_t 의 계산에 대한 내용이다. 가중치벡터는 제어기로부터 나온 실수 벡터를 가지고 메모리 내 벡터들 중 결과로써 적절한 위치를 특정하여 사상시켜 주는 기능을 수행한다. 이러한 가중치 벡터는 두 가지 주소지정 매커니즘을 고려하여 그 값이 계산된다. 첫 번째 매커니즘은 내용기반 주소지정이다. 이 매커니즘의 경우 제어기로부터 나온 실수벡터 값과 현재 메모리 내의 실수 벡터 중 가장 유사한 벡터의 위치에 헤드의 포커스를 집중시켜 주는 방법이다. 이러한 내용기반 주소지정의 장점은 메모리에 저장된 내용을 간단하게 그대로 불러올 수 있다는 것이다. 하지만 이러한 내용기반 주소지정이 모든 문제에 잘 적용되는 것은 아니며 이때 사용하는 기법이 위치기반 주소지정 매커니즘 이다. 대표적인 예를 하나 들면, 산술연산을 들 수 있다. 산술 연산의 경우 임의의 두 변수를 선택하여 연산을 수행하는데, 이때 변수의 내용을 보고 메모리 내의 유사 도를 비교하는 등의 번거로움 없이 단순히 수가 저장된 위치에 직접 접근하는 것이 효율적이다. 따라서 두 가지의 경우 각각의 장점이 있으므로 두 매커니즘을 적절히 조합해서 가중치 갱신에 사용하여야 하며 이를 위해서 그림 9.4 과 같은 다섯 가지의 변수를 추가적으로 고려해야 한다. 각각의 헤드마다 5 개의 변수들이 필요하며, 그림 9.4 의 각각의 박스는 1 층짜리 인공 신경망을 의미한다. 또한 그림 9.5 는 앞서 설명한 주소지정 매커니즘의 흐름을 개략화 한 것이다.

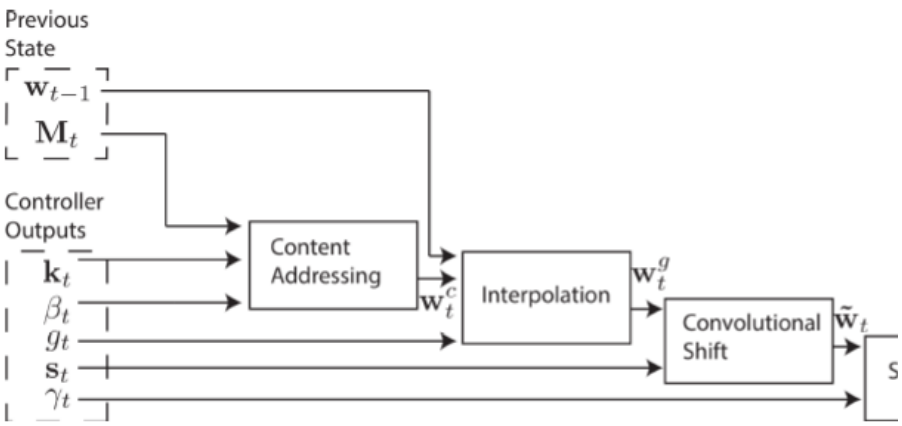


그림 9.5 주소지정 매커니즘의 흐름도(Siegelmann, 1992).

먼저 내용기반 주조지정 단계의 수식은 다음과 같다.

$$w_t^i(i) \leftarrow \frac{\exp\left(\beta_t K\left[k_t, M_t(i)\right]\right)}{\sum_j \exp\left(\beta_t K\left[k_t, M_t(j)\right]\right)}$$

읽기 또는 쓰기헤더는 먼저 M 길이의 키 벡터 k_t 를 생성시킨다. 이후 집중도를 증폭시키기 위한 키 강도 β_t 와 함께 현재의 메모리 M_t 내에서 k_t 와 가장 비슷한 벡터를 찾아 곱한 후 정규화를 시킨다. 이때 유사 도의 척도 K 는 코사인 유사 도를 사용하며 다음과 같다.

$$K\left[u, v\right] = \frac{u \cdot v}{\|u\| \cdot \|v\|}$$

다음으로는 위치기반 주소지정에 대해서 살펴보자. 위치기반 주소지정은 보간, 회선, 샤프닝으로 이루어져 있다. 먼저 보간은 이전 시간 단계에서 나온 가중치 w_{t-1} 와 현재 내용기반 주소지정으로부터 나온 가중치 w_t^c 를 0~1 사이의 값을 가지는 보간 게이트 g_t 를 사용하여 적당히 혼합시키는 역할을 하며 다음 수식과 같이 정의된다.

$$w_t^g \leftarrow g_t w_t^c + \left(1 - g_t\right) w_{t-1}$$

보간 이후 헤더는 이동 가중치 s_t 를 생성시킨다. 이후 생성된 이동 가중치에 의해 회선이동 되는 식은 다음과 같다.

$$\tilde{w}_t(i) \leftarrow \sum_{j=0}^{N-1} w_t^g\left(j\right) s_t\left(i-j\right)$$

이후 헤드에서 생성되는 샤프닝 가중치 $\gamma_t \geq 1$ 에 의해 최종적으로 샤프닝 됨으로써 주소지정이 완료된다.

$$w_t(i) \leftarrow \frac{\tilde{w}_t(i)^{\gamma_t}}{\sum_j \tilde{w}_t(j)^{\gamma_t}}$$

지정된 주소는 이후 외부 메모리에 접근하는 데 사용된다. 그림 9.6 는 이러한 가중치 계산의 전반적인 흐름을 도식화 한다.

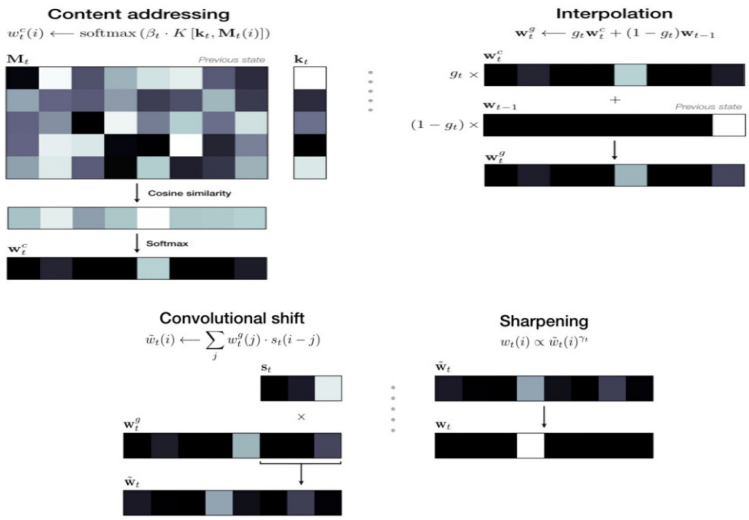


그림 9.6 가중치 계산의 흐름 예시

(그림 출처: https://medium.com/snips-ai/ntm-lasagne-a-library-for-neural-turing-machines-in-lasagne-2cdce6837315#.r921877sp)

9.2 메모리넷

메모리넷 모델은 인간의 기억 기제를 모사하여 컴퓨터의 메모리 장치를 장기 메모리로 활용한다. 장기 메모리는 입력 데이터가 들어옴에 따라 읽혀지거나 쓰여질 수 있고, 추론은 장기 메모리로부터 입력과 연관된 특정 메모리를 회상함으로써 이루어진다. 이러한 메모리 모델이 고안된 배경은, 기존 딥러닝 모델의 기억 용량에는 한계가 있었기 때문이다. 예를 들어, 순환 신경망은 순차적 입력을 기억하기 위해서 은닉 유닛을 활용한다(Hochreiter & Schmidhuber, 1997). 이 모델은 입력 정보가 순차적으로 들어올 때마다 고정된 크기의 은닉 유닛의 값을 업데이트 시키면서 기억을 하는데 이 때 두 가지 문제가 발생한다. 1) 입력의 길이가 길어질수록 은닉 유닛의 크기도 커져야 하고 2) 은닉 유닛의 업데이트가 이루어질수록 입력 순서의 앞 부분에 위치한 정보가 손실된다. 메모리넷 모델은 순차적 입력의 모든 정보를 하나의 고정된 크기의 은닉 유닛에

인코딩시키기 보다는 입력이 들어오는 각 시간 순서마다 히든 유닛을 만들어서 장기 메모리에 저장을 한다. 그림 9.7 은 메모리넷 모델의 구조도를 나타낸다. 모델은 다음과 같이 네 가지 모듈로 구성되어 있다(Weston et el., 2014).

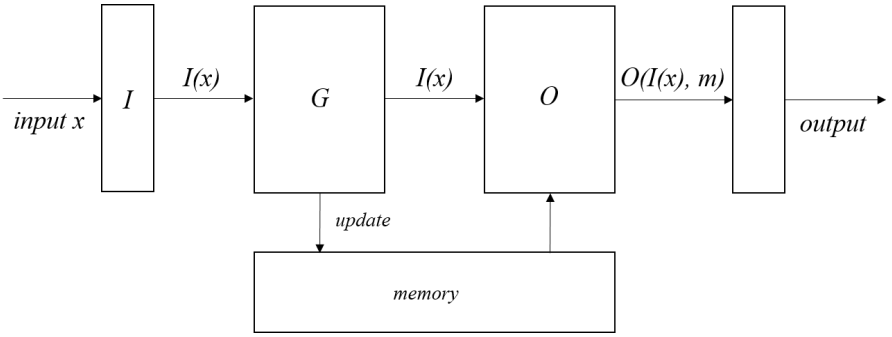


그림 9.7 메모리넷 모델의 구조도

I: (입력 특징 모듈) – 들어오는 입력 x를 특징 벡터 $I(x)$로 변환시킨다. I의 한 예로 일반적인 전처리 과정이 있다 (예> 파싱, 엔티티 분석 등). 또한 임베딩 행렬을 통한 분산 정보 표현도 가능하다.
G: (일반화 모듈) – 새로운 입력이 주어졌을 때 오래된 메모리를 업데이트하거나 새로 추가한다. 가장 간단한 형식의 G는 $I(x)$를 메모리의 한 슬롯에 저장을 시키는 것이다. $m_{H(x)} = I(x)$. 이 때, $H(.)$는 슬롯을 선택하는 함수이다. G는 메모리 m의 인덱스 $H(x)$를 업데이트하지만 메모리의 나머지 부분은 그대로 남게 된다.
O: (출력 특징 모듈) – 입력과 현재 메모리 상태를 바탕으로 새로운 출력을 생성한다. 출력은 특징 공간상에서 표현된다.
R: (응답 모듈) – 특징 공간 상에서 표현된 출력을 단어와 같이 사람이 이해할 수 있는 포맷으로 변환시킨다.

입력 x (예> 단어) 가 주어졌을 때 모델의 흐름은 다음과 같다.

1. x를 특징 벡터 $I(x)$로 변환한다.
2. 새로운 입력에 따라서 메모리 m_i를 업데이트시킨다. $m_i = G(m_i, I(x), m)$.
3. 입력과 메모리 상황에 따라 출력 특징 o를 계산한다. $o=(I(x), m)$.
4. 마지막으로, o를 최종 응답 r로 디코딩한다. $r=R(o)$.

이러한 과정은 훈련 시간과 테스트 시간에 모두 적용이 되나 테스트 시간에는 모델의 파라미터 I,G,O 그리고 R은 변경되지 않는다. 파라미터

I, G, O 그리고 R은 적절한 기계학습 모델로 선택할 수 있다. 아래부터는 신경망을 활용한 기본 모델을 설명한다.

I 모듈이 텍스트 문장들을 입력으로 받는다고 가정하자. 이 텍스트들은 메모리 상에서 다음으로 가능한 슬롯에 저장이 된다. $H(x)$는 다음으로 비어 있는 메모리 슬롯 N을 반환하고 $m_N = x$, $N = N + 1$ 이 된다. G모듈은 따라서 새로운 입력을 메모리에 저장하고 다른 저장되어 있는 메모리는 변경하지 않는다. 추론은 O와 R 모듈에서 일어난다. O 모듈은 x가 주어졌을 때 k개의 관련 메모리들을 찾아서 출력을 생성한다. 예시로 k는 2 를 사용하나 더 큰 값의 k가 사용될 수 있다. k가 1 일 때, 가장 높은 매칭 점수를 얻는 관련 메모리가 선택된다.

$$o_1 = O_1(x, m) = \arg\max_{i=1,\ldots,N} s_O(x, m_i)$$

이 때, s_O는 문장 x와 메모리 m_i의 쌍을 비교하고 점수를 매기는 함수이다. k는 2 인 경우, 위에서 선택한 관련 메모리와 입력 문장을 바탕으로 두 번째 관련 메모리를 찾는다.

$$o_2 = O_2(x, m) = \arg\max_{i=1,\ldots,N} s_O([x, m_{o_1}], m_i)$$

관련 후보 메모리 m_i는 입력과 첫 번째 관련 메모리에 기반하여 점수가 매겨진다. [.]는 두 개의 벡터를 합치는 연산이고 간단한 예로 더하기 연산이 될 수 있다. 출력 o는 $[x, m_{o1}, m_{o2}]$이고 R모듈의 입력이 된다. R모듈은 텍스트 응답 r을 생성한다. 가장 간단한 응답은 가장 마지막으로 선택된 관련 메모리 m_{ok}를 반환하는 것이다. 문장 생성을 하려면 이 모듈에서 RNN을 사용할 수 있다. 출력인 한 단어일 경우 후보 단어들에 점수를 매겨서 선택할 수 있다.

$$r = \arg\max_{w \in W} s_R([x, m_{o_1}, m_{o_2}], w)$$

W는 사전에 있는 모든 단어의 집합이고, s_R은 매치에 점수를 매기는 함수이다.

O와 R의 예로, 질의 x='Where is the milk now'가 주어졌을 때, O모듈은 x를 모든 메모리와 비교를 한 뒤 점수를 매긴다. 이 때, 메모리는 예전에

모델이 봤던 문장들이 된다. 이 때, 가장 높은 점수를 얻는 메모리 m_{o1}가 'Joe left the milk'를 저장하고 있다면, $[x,m_{o1}]$을 바탕으로 두 번째 관련 메모리를 찾는다. m_{o2}는 'Joe travelled to the office'가 될 수 있다. 마지막으로 R모듈은 $[x,m_{o1},m_{o2}]$이 주어졌을 때 최종 응답 r='office'를 찾을 수 있다.

신경망을 사용하는 함수 s_O와 s_R은 다음과 같은 형식이 될 수 있다.

$$s(x,y)=\phi_x(x)^{\mathrm{T}}U^{\mathrm{T}}U\phi_y(y)$$

U는 n x D 크기의 행렬이고 D는 특징 개수, n은 임베딩 크기를 의미한다. ϕ_x 와 ϕ_y는 입력 문장을 D크기의 특징 공간에 매핑 시켜주는 함수이다. 간단한 특징 공간은 단어주머니(bag-of-words)를 사용하는 것이다.

학습은 훈련데이터에 주어진 입력과 최종 응답으로 감독학습을 할 수 있고, 최종 응답과 관련된 문장 정보는 주어진다. 즉, 훈련 시간 동안에는 위의 맥스 함수들의 정답을 알고 있다. 훈련은 마진 기반 랭킹 함수와 확률 그라디언트 하강(stochastic gradient descent, SGD)를 사용한다. 주어진 입력 x와 출력 r과 관련 메모리 m_{o1}과 m_{o2}가 있을 때, 모델 파라미터 U_O와 U_R을 최소화시키는 방향으로 학습이 이뤄진다.

$$\sum_{\bar{f}\neq m_{o1}}\max(0,\gamma-s_O(x,m_{o1})+s_O(x,\bar{f}))+$$
$$\sum_{\bar{f}\neq m_{o2}}\max(0,\gamma-s_O([x,m_{o1}],m_{o2})+s_O([x,m_{o1}],\bar{f}'))+$$
$$\sum_{\bar{r}\neq r}\max(0,\gamma-s_R([x,m_{o1},m_{o2}],r)+s_R([x,m_{o1},m_{o2}],\bar{r}))$$

$\bar{f}$, $\bar{f}'$ 와 $\bar{r}$ 는 정답 레이블이 아닌 다른 선택 값들이고 γ 는 마진이다. R모듈에서 RNN 을 사용하여 문장을 생성하는 경우, 위의 식을 일반적인 언어모델에서 사용하는 일반적인 로그 우도함수를 사용할 수 있다.

9.2.1 종단간 처리 메모리넷 (MemN2N)

MemN2N 은 메모리 모델의 한 가지로 종단간 처리방식으로 학습이 되어 훈련 과정에서 추가적인 감독이 필요하지 않다(Sukhbaatar et el., 2014).

또한 집중 기작을 갖고 있는 특징이 있다는 점에서 (Weston et el., 2014)와 다르다.
모델은 이산 입력 x_1, …, x_n을 받아 메모리에 저장하고, 질의 q에 대해 정답 a를 출력으로 내놓는다. x_i, q, 그리고 a는 사전에 있는 단어 집합 V의 기호를 포함한다. 모델은 모든 x를 고정된 크기의 메모리에 저장하고 x와 q에 대한 분산 표현을 계산한다. 분산 표현은 여러 단계에 걸쳐 계산되며 최종 출력 a를 생성하는데 사용된다. 훈련 과정 동안 에러값은 여러 메모리 접근 단계에 걸쳐 역전파된다.

이제 모델이 단일 메모리 접근 연산을 할 때 과정을 설명하겠다.
입력 x_1, …, x_n이 주어지면 메모리에 저장한다. 모든 $\{x_i\}$는 x_i를 분산 벡터 공간 상에 임베딩 시킴으로써 d 크기의 메모리 벡터 $\{m_i\}$로 변형이 된다. 이 때 임베딩 행렬을 A라 하고, A는 d x V 크기를 가진다. 질의 q는 크기는 같지만 다른 임베딩 행렁 B를 사용해 분산 벡터 u로 표현된다. 이 때, 임베딩 공간에서 u와 각 메모리 m_i를 비교 매칭하여 점수를 계산한다.

$$p_i = \text{Softmax}(u^{\text{T}} m_i)$$

이와 같이 계산한 p는 질의 q에 대한 각 메모리 벡터의 확률 값으로 볼 수 있다. 그리고 각 x_i는 또 다른 임베딩 행렬 C에 의해 분산 벡터 c_i로 표현된다. q에 대해 정리된 메모리 벡터는 확률 값이 가중치된 c_i의 합이다.

$$o = \sum_i p_i c_i$$

이처럼 모델은 집중 기작을 가지고 있고, 함수의 입력에서 출력까지 연속적이기 때문에 역전파 과정을 통해 파라미터 값들을 학습할 수 있다. 단일 메모리 접근 연산인 경우, 메모리 출력 o와 질의 u의 합은 마지막 파라미터 행렬 W의 입력으로 들어가고 소프트맥스를 통해 예측 레이블을 계산한다.

$$\hat{a} = \text{Softmax}(W(o+u))$$

전제 모델은 그림 9.7 에 나타나 있다. 훈련 과정동안 세 개의 임베딩 행렬 A,B,C와 W는 예측 레이블 $\hat{a}$와 정답 레이블 a 사이의 크로스 엔트로피

손실을 줄이는 방향으로 한꺼번에 학습이 된다. 훈련 과정은 SGD 로 학습이 된다.

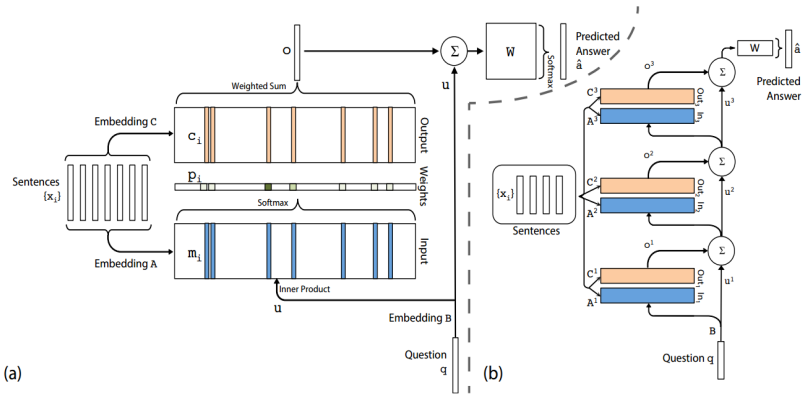


그림 9.8 (a) 단일 메모리 접근 연산 시 모델의 모습 (b) 세 번의 메모리 접근 연산 시 모델의 모습 (Sukhbaatar et el., 2014)

이제 모델을 K번 메모리 접근 연산으로 확장해보겠다. 메모리 층은 다음과 같은 방식으로 확장된다.

- 첫 번째 메모리 접근 이후 다음 메모리 층의 입력은 k번째 메모리 층의 출력 o_k와 입력 u_k의 합이다.

$$u^{k+1} = u^k + o^k$$

- 각 층은는 입력 $\{x_i\}$를 임베딩시키기 위한 서로 다른 임베딩 행렬 A_k, C_k를 갖는다. 하지만 이 행렬들은 모델의 파라미터 개수를 줄이기 위해 아래에 설명된 대로 변형될 수 있다.

- 모델의 가장 마지막에서 행렬 W의 입력은 마지막 메모리 층의 출력과 입력을 합친다.

$$\hat{a} = \text{Softmax}(Wu^{K+1}) = \text{Softmax}(W(o^k + u^k))$$

모델의 파라미터를 줄이기 위해 다음과 같은 방법을 사용할 수 있다.

1. 인접 방식: 메모리 층의 임베딩 C는 그 다음 층의 임베딩 A와 같게 한다 ($A_{k+1}=C_k$). 그리고 정답 예측 행렬 W는 마지막 메모리 층의

임베딩 행렬 C와 같게 한다 ($W^T=C_K$). 질의 임베딩 행렬은 첫 번째 층의 임베딩 A와 같게 한다 ($B=A_1$).

2. RNN 방식: 임베딩 행렬 A와 C를 모든 층에 걸쳐 동일하게 한다 ($A_1=A_2=\cdots=A_K$, $C_1=C_2=\cdots C_K$). 그리고 선형 매핑 H를 메모리 층 간에 적용하는 방법도 있다 ($u_{K+1}=u_k+o_k$). 이러한 매핑은 나머지 파라미터에도 적용된다.

세 번 메모리 접근 연산 시 모델의 모습이 그림 9.8(b)에 나타나 있다. 모델은 전형적인 RNN 으로 볼 수 가 있고, 이 때 RNN 의 출력은 내부, 외부 출력으로 나뉜다고 볼 수 있다. 내부 출력을 계산하는 것은 메모리 계산과 관련이 있고 외부 출력을 계산하는 것은 정답을 예측하는 것으로 볼 수 있다. RNN 의 관점에서 그림 9.8(b)의 u는 은닉 유닛이며 모델은 내부 출력 p를 임베딩 행렬 A 를 통해 계산한다. 이 때, p는 또한 집중 기작으로 생성한 집중 값이다. 모델은 p와 C를 통해 은닉 유닛을 업데이트 한다. 하지만 일반적인 RNN 과 달리 K번의 단계 동안 메모리에 저장되어 있는 출력들이 관여하게 된다.

9.3 생성 대립넷

생성 대립넷은 데이터의 분포를 학습하는 생성 모델의 일종이다(Goodfellow et el., 2014). 학습된 데이터 분포로부터 임의의 데이터를 생성할 수 있기 때문에 생성이라는 용어가 붙는다. 이 넷 또한 미분이 가능한 인공신경망으로 구성되어 있는 딥러닝 모델이다.

생성 대립넷에는 두 넷이 존재하는데 하나는 생성 넷이고 다른 하나는 분류 넷이다. 학습에는 둘이 적대적으로 대결하는 게임 이론에 기반한 시나리오를 따른다. 생성 넷은 사전 분포로부터 임의로 표집된 $\mathbf{z}$ 로부터 데이터 $\mathbf{x} = G(\mathbf{z})$ 를 생성한다. 분류 넷은 생성 넷이 생성된 데이터와 실제 데이터를 구분하려 한다. 이 넷이 출력하는 값 $D(\mathbf{x})$ 는 $\mathbf{x}$ 가 실제일 확률이며 1 과 0 사이의 값이다. 반대로 생성 넷은 분류 넷을 속일 수 있는 실제와 같은 데이터를 생성하려 한다. 두 넷은 이렇게 서로 속고 속이는 과정 속에서 대립적으로 학습되는 궁극적으로 실제 데이터 분포에 알맞은 데이터를 생성하기에 이른다. 충분한 양의 학습 데이터가 있으면 생성 넷은 단순히 학습 데이터를 외우기보단 일반화를 추구한다.

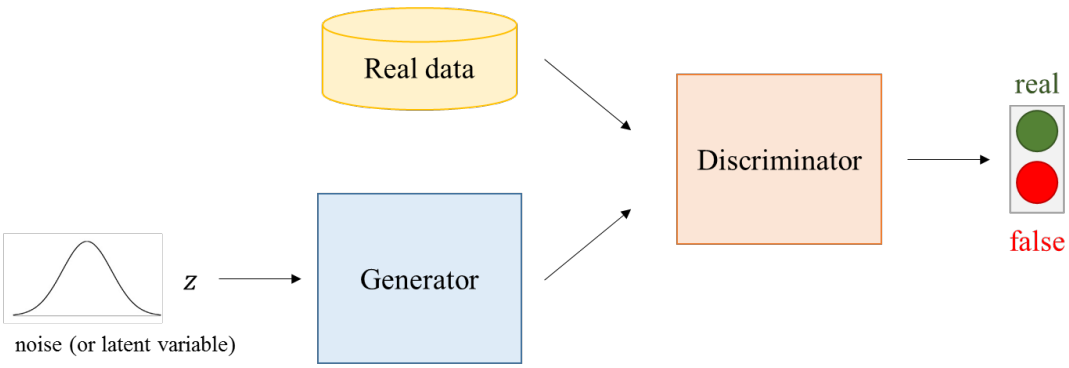


그림 9.9 생성 대립넷의 모식도

실제 데이터의 확률 분포를 p_{data} 라 하고 생성 넷이 생성하는 데이터의 확률 분포를 p_{model} 이라고 할 때, 생성 대립넷은 파라미터 학습 통해 최적의 $v(G,D)$ 를 형성하는 G 와 D 를 찾는다.

$$v(G,D)=\min_G \max_D E_{x\sim p_{data}}[\log D(\mathbf{x})]+E_{x\sim p_{\text{model}}}[\log(1-D(\mathbf{x}))]$$

여기서 $\log D(\mathbf{x})$ 는 $D(\mathbf{x})$ 가 1 일 때, 최대가 되고 반대로 0 에 가까워지면 음의 무한대로 발산한다. 따라서 G 가 고정된 상태에서 $v(G,D)$ 를 최적화하기 위해서는 실제 데이터에 대해서는 $D(\mathbf{x})$ 가 1 에 가까워야 하고 생성 넷이 생성한 데이터는 $D(\mathbf{x})$ 가 0 에 가까워야 한다. 반대로 D 가 고정된 상태에서는 $E_{x\sim p_{data}}[\log D(\mathbf{x})]$ 가 G 와 무관한 식이므로 생략되고 모델에 대한 $D(\mathbf{x})$ 가 1 에 가까울 때 $v(G,D)$ 가 최적화된다. 두 넷이 오랜 시간 대립적으로 학습되면서 최종적으로 p_{model} 과 p_{data} 이 같아지는 경우에 $v(G,D)$ 가 수렴이 되고 $D(\mathbf{x})$ 가 항상 $\frac{1}{2}$ 이 된다. 이 순간에는 D 가 더 이상 가짜와 진짜를 구분할 수 없는 상황이다. 모든 학습은 $v(G,D)$ 값을 감소(증가)시키는 방향으로 파라미터를 변화시키는 경사 하(상)강법 방식으로 진행된다.

[생성 대립넷 학습 알고리듬]

for number of training iterations **do**

Sample minibatch of m noise samples $\{z^{(1)},...,z^{(m)}\}$ from noise prior.

Sample minibatch of m examples $\{x^{(1)},...,x^{(m)}\}$ from real data.

Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^{m}[\log D(x^{(i)})+\log(1-D(G(z^{(i)})))]$$

Sample minibatch of m noise samples $\{z^{(1)},...,z^{(m)}\}$ from noise prior.

Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^{m}[\log(1-D(G(z^{(i)})))]$$

end for

이러한 학습 방법은 기존의 생성 모델과 비교하여 여러 장점이 있다. 첫째는 볼쯔만머신과 같은 확률 모델에서는 사후확률을 근사하기 위해 마르코브 체인 몬테 카를로 표집법을 사용하는데 이는 고차원을 다루는

경우 많은 학습 시간을 요구하지만 생성 대립넷은 경사 하강법을 사용하여 신속하고 직접적인 학습을 진행한다. 둘째는 베이지안 통계적 추론에 근사한 대체 함수를 사용하지 않는다는 점이다. 대체 함수도 미분을 통해 학습을 할 수 있지만 어디까지나 최적값의 하한이기 때문에 부정확하다. 셋째는 기본적으로 인공신경망으로 D와 G의 모델을 구성하므로 이미지 데이터에 적절한 컨볼루션 인공신경망과 같은 강력한 모델을 자유롭게 활용할 수 있다. 특히 배치 정규화 기법과 깊은 컨볼루션 인공신경망으로 구성된 생성 대립넷은 복잡한 이미지 데이터에서도 품질이 높은 데이터를 생성한다. 더욱이 인공신경망의 복잡도를 고려한다면 임의의 노이즈로부터 생성된 데이터들이 대부분 실제 데이터와 비슷할 수 있는 것은 상당히 강력한 학습 능력이다.

그림 9.10 딥 컨볼루션 생성 대립넷으로 생성한 이미지 데이터(Radford et el., 2015)

이론적으로 만약 생성 넷이 충분한 표현력을 가지고 있고 데이터가 충분하다면 모델이 생성하는 데이터의 확률 분포가 실제 데이터의 확률 분포에 수렴한다고 증명되었다. 그러나 실제로는 그러한 가정이 현실적이지 않으며 하이퍼매개변수에 따라 학습이 불안정하거나 전혀 안 되는 경우도 많다. 또한 전반적인 학습 양상이 넷 구조에 크게 영향을 받는다.

이러한 문제점을 보완하기 위해 생성 대립넷을 개선하는 방법들이 고안되었다. 가장 대표적으로 딥 컨볼루션이라는 수식어가 앞에 붙는 모델로 각종 딥러닝 튜닝 기술을 적극적으로 활용하여 학습 속도와

안정성을 크게 개선하였다(Radford et el., 2015). 처음 생성 대립넷이 제안되었을 당시에는 손으로 쓴 숫자 데이터 정도에서 데모를 보였고 그 이외에 더 복잡한 데이터에서는 육안으로 인지하기 어려운 낮은 품질의 데이터가 생성되었다. 반면 딥 컨볼루션 생성 대립넷은 얼굴 또는 꽃 사진처럼 복잡한 이미지로도 학습이 잘 된다. 이 모델은 네 개 이상의 컨볼루션 층으로 분류 넷을 구성하고, 비슷한 수의 디컨볼루션 층으로 생성 넷을 구성하였다. 디컨볼루션은 컨볼루션의 흐름이 거울에 반사된 것과 같은 정반대 형태이고 이에 따라 필터의 크기가 역으로 커진다. 이 모델에 사용되는 컨볼루션은 필터 추출 간격을 2 로 하고 풀링을 생략하여 과대적합화 문제를 완화하였다. 또한 배치 정규화를 적용하여 학습의 속도를 배로 높였고 아담(adam) 최적화 기법 등이 응용되었다.

넷의 구조를 개조해서 생성과 분류의 조건을 부여할 수 있는 방법도 개발되었다(Reed et el., 2016). 예를 들어 숫자 그림을 생성하는 모델에 특정 숫자의 그림만 생성하도록 제약을 걸 수 있다. 이러한 모델에는 각 층하고 노이즈 벡터에 조건에 해당되는 조건 벡터가 결합된다. 이 벡터에 따라 생성 넷은 생성할 대상을 결정하고 분류 넷 특정 대상만 구분한다. 조건 벡터를 데이터를 설명하는 문장 수준으로 더욱 고도화된 모델도 개발되었다. 이 모델은 주어진 설명에 부합하는 사진을 생성하기도 한다.

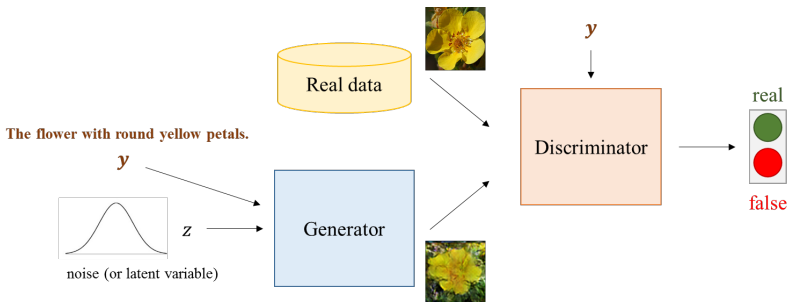


그림 9.11 조건을 부여한 생성 대립넷 모식도

생성 모델은 단순히 어떤 데이터를 생성하는 기능을 가졌다는 해석을 넘어서 데이터의 일반화된 분포를 한정된 데이터를 가지고 학습하는 모델로 이해되어야 한다. $\mathbf{z}$ 가 형성하는 공간은 $\mathbf{z}$ 로부터 생성되는 가짜

데이터 공간으로 투사되는데 이 공간이 실제 데이터 공간과 비슷해야 실제와 같은 데이터가 생성된다. 이러한 특성은 분류 모델과 큰 차이가 있다. 분류 모델은 서로 다른 클래스를 구분하기 위해 데이터 공간에서 구분 경계선을 찾는다. 이러한 근본적인 차이로 생성 모델은 분류 모델로는 풀기 어려운 여러 문제를 해결하는데 응용된다. 예를 들어 어떤 이미지에 모자이크를 제거하는 문제를 해결하기 위해서는 모자이크가 없는 부분을 조건으로 가려진 부분의 일반적 분포를 이해해야 한다. 생성 대립넷은 깊은 컨볼루션 인공신경망을 이용하여 해상도 보정, 모자이크 제거, 동영상 생성 등의 문제에 응용될 수 있다.

우리가 현실로부터 보는 영상은 그 구조가 고도화되어 있고 복잡하며 다양하지만 사람의 뇌는 신기하게도 그러한 영상들로부터 추상화된 개념을 쉽게 이해한다. 한 예로 어떤 이미지에 중간 부분이 일부 가려져 있어도 주변의 정보들로부터 가려진 부분이 어떻게 생겼을지 순식간에 상상한다. 생성 대립넷 또한 이러한 문제를 해결하는데 유용하게 사용될 수 있다(Pathak et el., 2016). 학습 방법은 원래 것과 거의 비슷한데 조건으로 특정 부분이 가려진 이미지가 주어진다. 가려진 이미지는 컨볼루션 인코더로 특징이 추출되고 이 특징이 생성 넷의 입력으로 들어간다. 생성 넷에서 출력된 이미지는 조건으로 주어진 이미지와 결합되어 가려진 부분만 생성된 것으로 채워진다. 이것이 마지막에 분류 넷의 입력으로 들어가게 된다. 기존의 목적함수에 추가적으로 가려진 부분을 채운 이미지와 실제 완전한 이미지와의 차이를 픽셀단위로 계산하는 오류 항이 있다. 이 항은 최종 이미지의 구체성과 정확성을 높이는 역할을 한다.

그림 9.12 생성 대립넷으로 빈 공간을 채우는 예시(Pathak et el., 2016)

초고해상도 영상처리는 흐린 이미지의 해상도를 보정하는 기술로 특히 저화질의 CCTV 영상을 보정하는데 널리 사용된다. 생성 대립넷을 이용하면 다른 딥러닝 기술과 비교하여 상당한 수준의 성능으로 이 기술을 구현할 수 있다(Ledig et el., 2016). 이때에는 저해상도 이미지가 생성 넷의 입력이 되고 분류 넷은 실제 고해상도 이미지인지 구분한다.

그림 9.13 생성 대립넷(가장 오른쪽)로 해상도 보정을 하는 예시. 가장 왼쪽에 있는 이미지의 해상도를 4배 줄인다(Ledig et el., 2016)

9.4 인코더-디코더 모델: 문장 대 문장

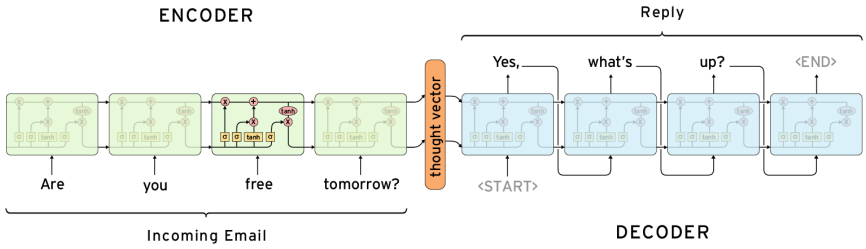


그림 9.14 인코더-디코더 모델

(그림 출처: https://research.googleblog.com/2015/11/computer-respond-to-this-email.html)

딥러닝은 학습의 모든 분야에서 뛰어난 성능을 보여왔지만 한계점이 존재했다. 그 중 대표적인 예가 자연어 처리 (Natural Language Processing) 분야에서 나타났다. 자연어 처리란 우리의 일상 생활 언어를 컴퓨터가 이해할 수 있도록 가공, 처리하는 분야인데 딥러닝은 입력과 출력의 1 대 1 연결 관계를 가지기 때문에 기계 번역, 대화 생성 등의 자연어 처리를 하기에 부적합하여 다양한 부가적인 처리를 해야했다. “Hard work will lead you to success” 라는 문장을 우리 말로 번역하는 예를 생각해보자. 기존 단어 단위의 딥러닝에서는 Hard / work / will / lead / you / to / success 의 7 개의 입력이 들어갔을 때, 근면한 / 일 / 앞으로 / 이끌다 / 당신을 / 으로 / 성공 과 같은 동일한 길이의 출력 결과를 만들어내고 보다 자연스러운 문장을 만드는 처리가 필요했다. 그러나 대표적인 인코더-디코더 모델인 문장 대 문장 모델 (sequence to sequence model) 혹은 문장 대 문장 프레임워크에서 이 문제를 해결하였다(Sutskeveret el., 2014).

문장 대 문장 모델은 연속적인 속성을 가진 데이터를 입력으로 받아, 연속적인 속성을 가진 데이터를 출력하는 모델이다. 그러나 일반적인 순환 신경망과는 차이가 있다.

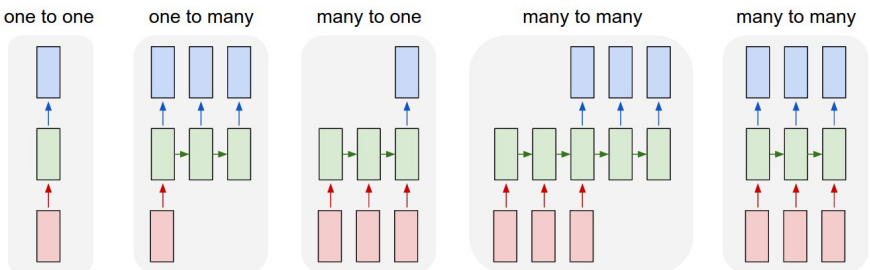


그림 9.15 순환 신경망의 다양한 구조

(그림 출처: http://karpathy.github.io/2015/05/21/rnn-effectiveness)

그전에 먼저 순환 신경망의 활용 방법을 입력-출력 간의 관계에 따라 분류하면 위의 그림과 같다. 하지만 우리가 다루고자 하는 자연어는 여러 개의 단어로 이루어져 있기 때문에 단일 대 단일(one to one), 단일 대 다(one to many), 다 대 다(many to one)는 문장을 입력으로 넣어 문장을 출력하는 기계 번역, 대화 생성 문제에는 적합하지 않다. 또한 다섯 번째 그림의 구조는 입력 문장과 출력 문장의 길이가 가변적인 자연어 처리에는 적합하지 않다. many to many 가 앞으로 설명하게 될 문장 대 문장의 구조를 간략화한 그림이다.

문장 대 문장은 앞서 설명한 순환 신경망 중 장단기 메모리를 활용한다. 기본적인 순환 신경망은 은닉 상태를 덮어쓰는데, 장단기 메모리는 은닉 상태를 업데이트 하는 구조를 가지고 있기 때문이다. 기본적인 순환 신경망 및 장단기 메모리의 자세한 내용은 7 장을 참고하기 바란다.

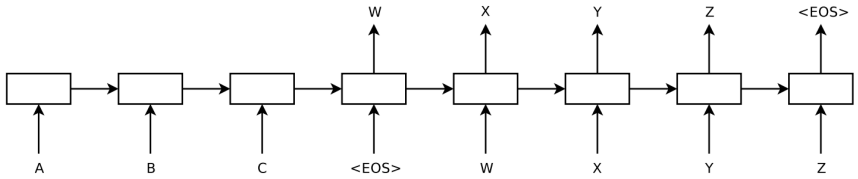


그림9.16 문장 대 문장 프레임워크(Sutskeveret el., 2014)

그림 9.16 를 통해 문장 대 문장 모델의 작동원리에 대해 설명해보겠다. 이때 입력은 “ABC”, 출력은 “WXYZ”, <EOS>는 문장의 끝을 나타내는 특수 기호이며, 각 네모 블록은 매 시간 t 마다의 장단기 메모리의 은닉층을 의미한다.

각 블록이 t=1 에서 t=7 에 해당한다고 가정했을 때, t=1 에서 t=3 까지 “ABC”의 문장은 각각 ‘A’, ‘B’, ‘C’의 단어 (혹은 청크) 단위로 쪼개어져 장단기 메모리의 입력 값으로 들어간다. t=4 에서는 <EOS>가 입력 값으로 들어가는데 이 부분부터 첫 출력으로 ‘W’가 나온다. t=5 에서는 t=4 일 때 출력 단어인 ‘W’를 다시 입력으로 넣어주어 ‘W’의 다음 단어인 ‘X’를 출력한다. 계속해서 t=n 출력을 t=n+1 의 입력으로 넣어주어 <EOS>가 나올 때까지 반복한다. 결론적으로는 “ABC”의 문장을 넣었을 때 “WXYZ”의 출력이 나오도록 로그 확률(log probability)를 감소하는 방향으로 학습을 하는 것이다.

전체적인 문장 대 문장 모델의 작동을 둘로 쪼개보면, “ABC”가 순차적으로 입력으로 들어간 (그림에서는 t=4 의 블록) 은닉층을 만드는 인코딩과 인코딩한 은닉층을 하나하나 풀어서 ‘W’, ‘X’, ‘Y’, ‘Z’의 단어를 만드는 디코딩의 두 과정이 있다.

이러한 아이디어를 가지고, Sutskever, I. 등(Sutskeveret el., 2014)은 3.4 억개의 불어 단어, 3 억개의 영어 단어로 구성된 WMT’14 English to French 데이터셋을 이용하여 기계번역에 활용하였다. 입력 언어는 f, 출력 언어는 e 로 표현하였을 때 이를 수식으로 표현하면 다음과 같다.

$$\hat{e} = \arg\max_e p(e \mid f) = \arg\max_e p(f \mid e)p(e)$$

번역 모델인 $p(f \mid e)$ 를 이용해 순차적인 영어 단어들을 생성하고, 언어 모델인 $p(e)$ 를 이용해 단어들을 적절한 영어 문장으로 만드는 것으로 해석할 수 있다.

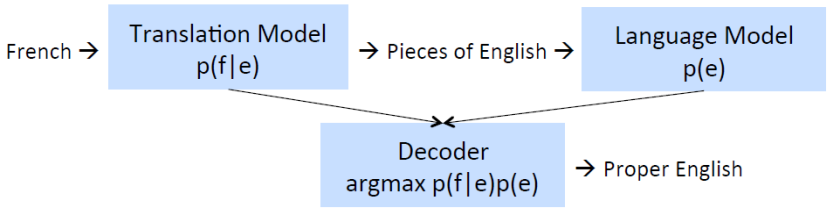


그림9.17 문장 대 문장의 세부 작동 과정

(그림 출처: SlideNotes from http://cs224d.stanford.edu/)

그리고 그는 보다 좋은 성능을 내기 위해 문장 대 문장 프레임워크를 더욱 깊게 쌓았다.

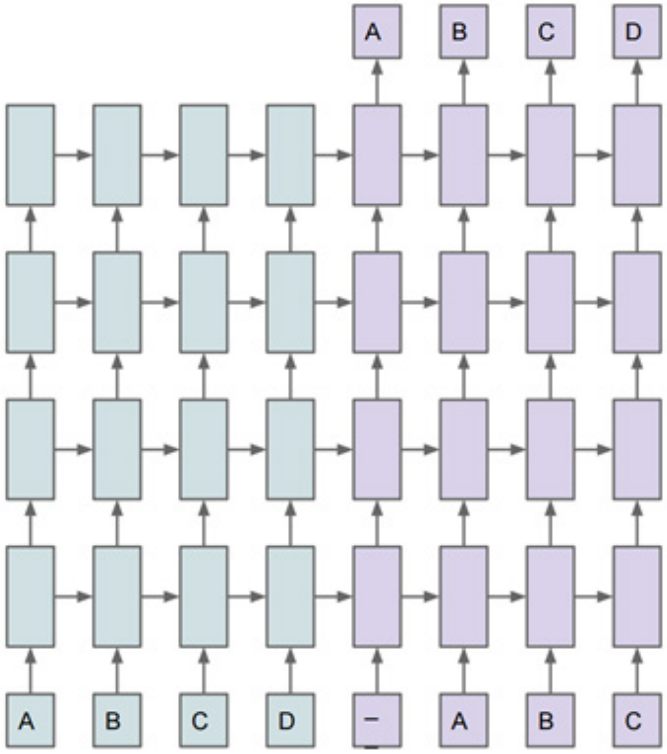


그림 9.18 기계번역을 위한 문장 대 문장 모델

(그림 출처: LectureNotes from NIP: Oral Session 4; https://www.microsoft.com/en-us/research/video/nips-oral-session-4-ilya-sutskever/)

Sutskever, I. 등. (Sutskeveret el., 2014)은 장단기 메모리를 그림 9.18 과 같은 구조로 쌓아 기계 번역 모델을 만들었고 기존 WMT'14 English to French 데이터셋의 state of art 와 거의 비슷한 성능을 보였다.

문장 대 문장 모델은 대화 생성 모델에 그대로 적용되었다. Vinyals, O., & Le, Q. (Vinyals & Le., 2015) 는 2015 년에 A neural conversational model 을 제안하였는데, 이는 기계 번역에 사용되었던 모델을 밑바탕으로 하여 사람과 대화가 가능한 모델을 만든 것이다. 그 모델 구조는 그림 9.19 과 같다.

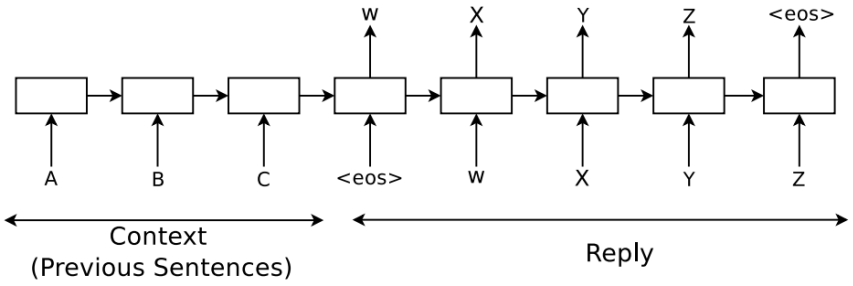


그림 9.19 대화형 모델을 위한 문장 대 문장 모델 (Vinyals & Le., 2015)

번역을 위한 데이터가 아닌 서로 다른 사람 A 와 B 가 서로 주고 받는 대화형 데이터를 이용하여 학습을 시켰다. 이 모델을 이용하여 총 두 가지 문제를 다루었는데, 첫 번째는 IT helpdesk 데이터셋을 이용하여 고객센터 질의응답을 할 수 있도록 학습했고, 두 번째는 영화 자막 파일을 수집하고 학습하여 일상적인 대화가 가능한 모델을 학습했다. 그러나 대화형 모델은 대화의 적절성을 판단하여 목적 함수를 설정하기 어렵고, 다양한 소스에서 수집한 데이터를 이용하여 학습을 하기 때문에 실제 사람과 같은 개성이 느껴지지 않는다는 단점이 있다.

문장 대 문장은 간단한 방법으로 순서를 가지고 있는 데이터를 효과적으로 처리할 수 있는 구조를 제공해주었다. 그리고 기존 자연어 처리에서 가변적인 길이를 처리하기 위해 하였던 수 많은 작업들을 단순화시키거나 할 필요가 없이 입력을 넣으면 출력이 나오는 종단간 처리의 편리함도 누릴 수 있게 해주었다. 문장 대 문장은 계속해서 활용되어 기계 번역, 음성 인식, 이미지 캡션 생성, 질문-대답 챌린지 등 특히 자연어 처리 분야에 많이 활용되고 있다.

9.5 딥 잔차넷

전통적인 시각 문제를 다룰 때에 이미지 정보에 대하여 경계선 탐지나 히스토그램, 또는 k 평균 클러스터링 이나 희소 코딩과 같은 전문적인 지식을 활용하여, 수작업으로 만든 특수한 요소들을 이용하는 방법을 많이 택하였다. 이미지 분류 문제를 다룬다고 가정하면, 이러한 전문적인 지식을 활용하여 요소들을 추가할 수록 분류기 전 단계의 깊이가 점점 깊어졌다. 하지만 이러한 방법은 전문가 지식이 필요하다는 점과 각 문제마다

수작업으로 만들어야 하는 요소들이 있다는 점 등의 이유로 좀 더 일반적인 접근 방법을 고민하게 만들었다.

딥러닝은 이러한 고민의 산물이며, 앞에서 살펴본 바와 같이 단순한 학습 층을 여러 층을 쌓아 학습할 수 있는 방법을 제시한다. 이것을 통해 오차역전파 알고리듬을 통해 전체 넷을 학습할 수 있으며 깊은 층의 넷은 더욱 풍부한 문제 해결 공간을 만들어주어 보다 높은 성능을 갖게 한다. 하지만 얼마나 깊은 넷이 가능할까?

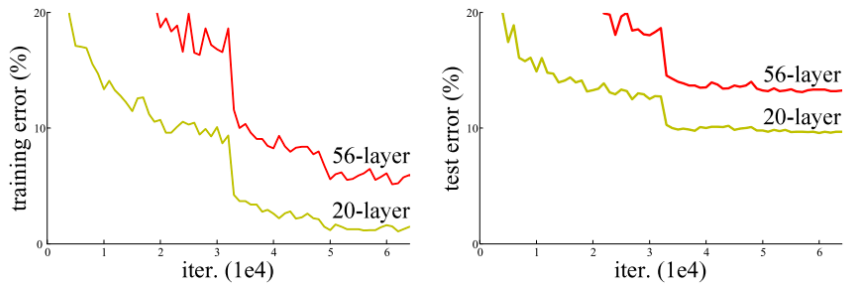


그림 9.20 일반적인 20층과 56층의 넷으로 CIFAR-10 데이터셋에 대한 학습 에러(왼쪽)와 테스트 에러(오른쪽) 곡선을 그려보면 위 그림과 같다. 깊이가 어느 정도 깊어지게 되면 학습 에러가 높아지고, 따라서, 테스트 에러도 크게 된다. (He et el., 2015)

더욱 많은 데이터, 더욱 천천히 작아지는 기울기를 만드는 ReLU 함수를 도입하면서 5 층 이상의 넷의 학습이 가능하게 되었다. 10 층 이상의 넷을 학습하는 데에는 학습 파라미터들의 초기화 전략, 배치 정규화 등의 기법이 크게 기여하였다. 30 층 이상과 100 층 이상과 같이 아주 깊은 넷을 구축하기 위한 방법으로는 몇 개의 층을 건너뛰는 연결을 만드는 방법을 사용한다. 이러한 넷을 바로 딥 잔차넷이라고라고 한다 (He et el., 2015, 2016).

먼저 딥 잔차넷은 보다 깊은 넷을 학습하기 위해 보수적인 접근 방법을 택하였다. 기존에 얕은 층의 넷의 층들을 복사한 후 사이 사이에 새로운 층을 추가하였다. 단, 최소한 같은 성능을 가질 수 있도록 새롭게 추가한 층은 입력 값과 같은 값을 내보내도록 파라미터를 초기화 하였다 (그림 9.23 가운데 열 참조). 하지만 여전히 깊은 층을 학습하는 어려움이 존재하므로 새로운 구조를 생각해야만 했다.

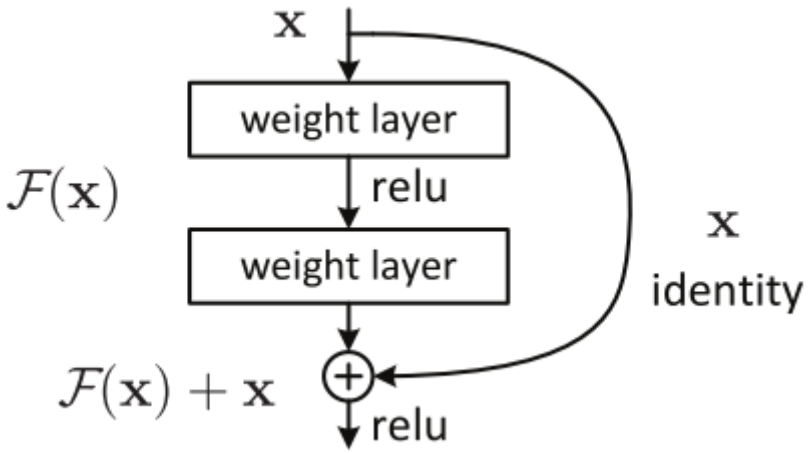


그림 9.21 딥 잔차넷의 잔차 학습 단위를 도식화 한다(He et el., 2015)

만일, 학습하고자 하는 맵핑이 H(x)이라면, 임의의 비선형 함수로 표현할 수 있는 넷은 H(x) – x 또는 F(x)를 학습하도록 하고, 입력 값 x를 F(x)에 더하는 형태로 넷을 만들어 결국 H(x)를 학습하도록 한다. 따라서 새롭게 추가된 넷이 기존의 학습된 넷의 성능을 최대한 유지시키면서 더 좋은 성능을 낼 수 있도록 집중할 수 있게 만들어준다.

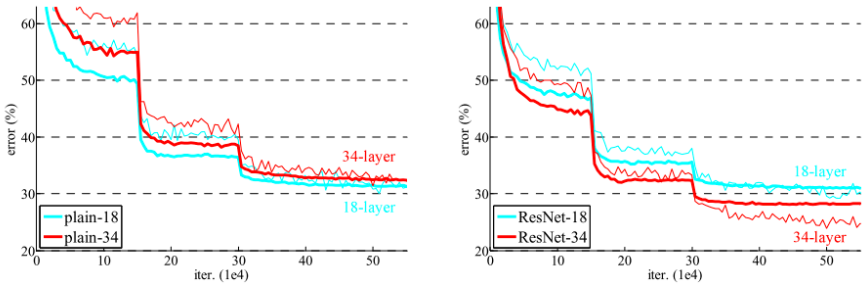
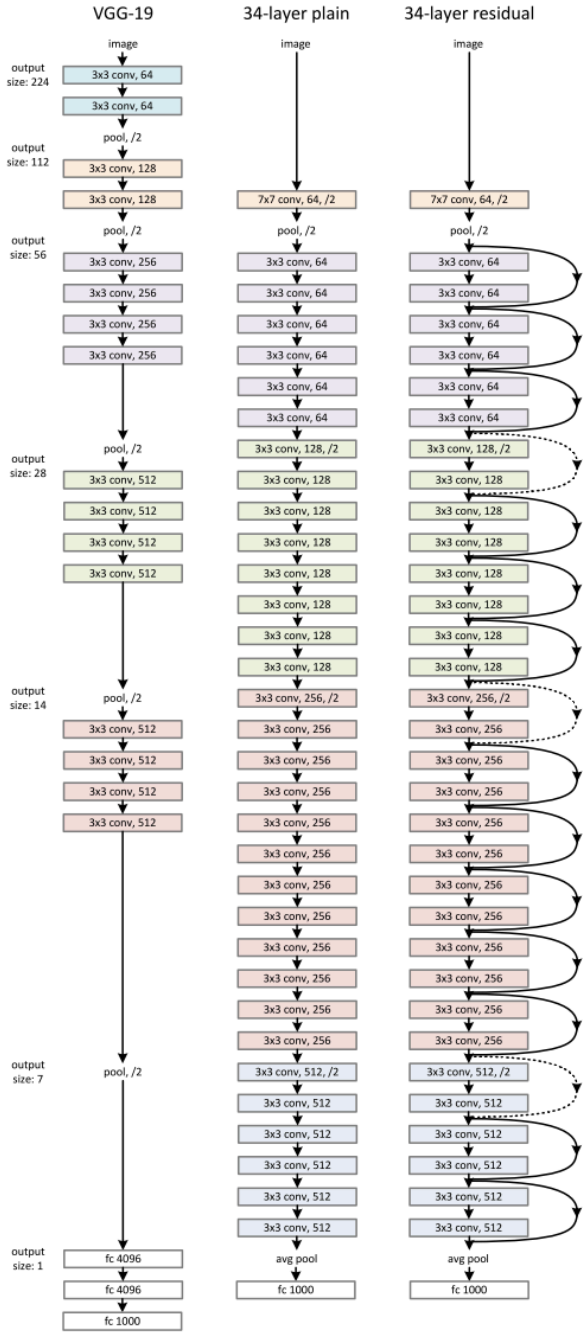


그림 9.22 이미지넷 데이터셋에 대한 실험 결과이다. 얇은 선은 학습 에러 곡선, 굵은 선은 검증 에러 곡선이다. 왼쪽 그림에서 깊이가 깊은 34층 넷은 에러가 18층 넷보다 큰 반면, 오른쪽 그림의 잔차넷에서는 18층 넷보다 34층 넷에서 보다 나은 성능을 보였다. (He et el., 2015)

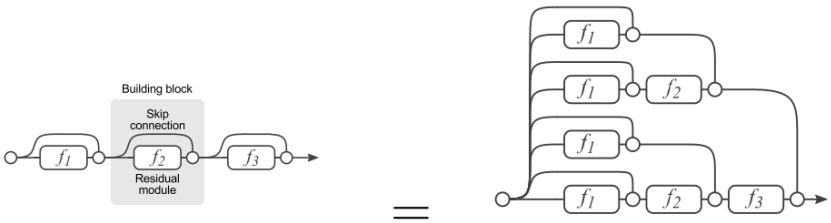
그림 9.22 은 딥 잔차넷을 이용하여 이미지넷 데이터셋을 학습하였을 때의 학습 곡선과 검증 곡선을 나타낸다. 지름 연결선을 이용한 깊은 넷은 상대적으로 얇은 넷보다 높은 성능을 가진다. 딥 잔차넷을 제안한 마이크로소프트 리서치의 Kaiming He 등은 2015 년의 이미지넷 분류,



탐지, 탐색과 COCO 대회의 탐지와 분리 분야에서 우승하였다.

그림 9.23 왼쪽 넷은 19층의 VGG-19 모델이며, 가운데의 넷은 34층의 깊은 넷을 도식화한다. 사이 사이 새로운 층이 추가되었다. 오른쪽 넷은 가운데 넷에서 지름 연결선들이 추가된 것을 보여준다. (He et el., 2015)

이후에 나온 후속 연구에서는 딥 잔차넷의 작동 원리가 깊이에 있는 것이 아니라고 새로이 주장한다. Veit 외 연구원들은 지름 연결선들이 정보가 전달되는 길을 기하급수적으로 늘려 상대적으로 얇은 층으로 구성된 여러 넷들을 덧셈의 형태로 암묵적인 앙상블 모델로 해석할 수 있다고 주장한다. (Veit et el., 2016)



(a) Conventional 3-block residual network (b) Unraveled view of (a)

그림 9.24 (a) 전형적인 3층의 잔차넷을 나타낸다. (b) 지름 연결선이 유발하는 정보 흐름의 경우들을 도식화하였다. (Veit et el., 2016)

그림 9.24 에서 관찰할 수 있듯이, N 개의 잔차 층으로 구성된 넷의 경우 입력과 출력을 연결하는 $O(2^n)$개의 암묵적인 연결선들로 해석할 수 있다.

앞에서 암묵적인 연결선들로 해석할 수 있는 암묵적 앙상블 모델이라고 일컫는 이유는 각 층에 위치할 수 있는 비선형 함수 때문이다. 따라서 모델에 대한 인위적인 조작 실험을 통해 이러한 가설을 검증하고자 한다.

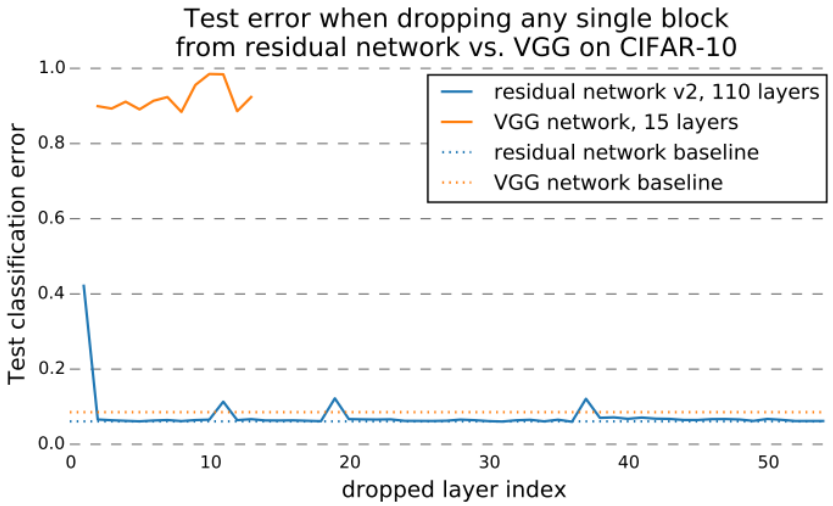
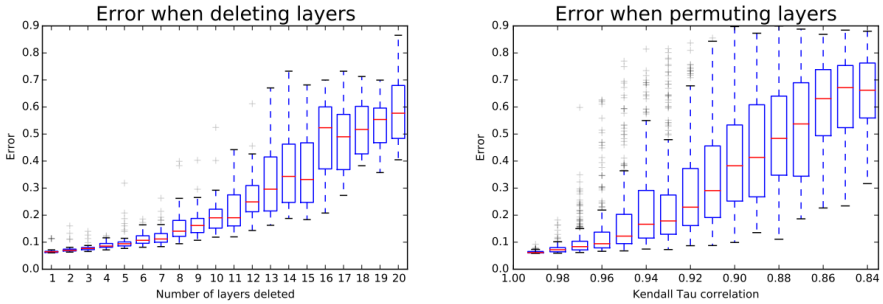


그림 9.25 파란 선은 110층의 잔차넷을, 주황색 선은 VGG 넷을 나타낸다. 학습 데이터는 CIFAR-10을 사용하였다. x축에 표시된 인덱스의 넷 층을 삭제하였을 때 성능의 변화를 관찰하였다. VGG 넷에서는 어느 위치에서든지 심각한 성능 손상을 가져왔지만, 잔차넷의 경우 거의 동일한 성능을 유지하였다. 단, 특징 벡터의 차원이 변경되는 구간에서 소폭의 성능 감소가 관찰된다. (Veit et el., 2016)

첫 번째 실험은 넷의 일부 층을 제거하여 성능의 변화를 관찰하였다(그림 9.25). VGG 넷의 경우 순차적인 넷이기 때문에 중간의 넷 층 하나를 제거하였을 때 즉시 심각한 성능 손상을 가져왔다. 하지만 잔차넷의 경우 첫 번째 층을 제외하면 거의 성능 변화를 관찰할 수 없었다. 이것은 앞의 해석에서 각 층이 암묵적인 앙상블 넷을 발생시킨다는 증거가 될 수 있다. 왜냐하면, 앙상블 학습은 이를 구성하는 한 넷이 빠져도 큰 성능 손상을 가져오지 않기 때문이다.



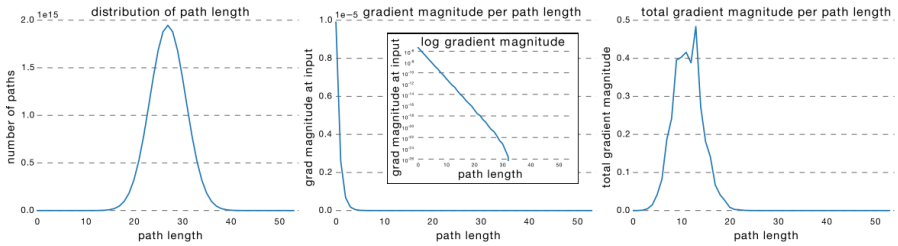
(a) (b)

그림 9.26 (a) 잔차넷에서 다수의 층을 제거하였을 때 에러가 점차적으로 변화하는 것을 관찰할 수 있다. (b) 잔차넷의 층들을 무작위로 뒤섞을 때의 에러 변화를 관찰하였다. 재정렬된 정도(Kendall Tau correlation)가 증가함에 따라 에러가 점차적으로 변한다. 이러한 결과는 앙상블 모델에서 쉽게 관측될 수 있는 모습이다. (Veit et el., 2016)

두 번째 실험은 잔차넷의 층들을 변화시켰을 때 앙상블 모델과 유사한 특성을 지니는지 관찰함으로써 앞의 가설을 확인하고자 한다. 그림 9.26(a)는 잔차넷에서 점차적으로 많은 층들을 제거하였을 때 나타나는 에러의 변화를 관찰한다. 제거된 층 수에 따라 점차적으로 에러 또한 증가함을 볼 수 있다. 일반적인 순차적 넷에서는 단 하나의 층을 제거하더라도 급격한 성능 하락을 경험한다.

그림 9.26(b)는 잔차넷의 층들을 무작위로 뒤섞을 때의 에러 변화를 관찰하고자 한다. 재정렬된 정도는 정량적으로 Kendall Tau 상관계수로 측정하며 x 축에서 나타난다. 재정렬된 정도가 증가함에 따라 에러가 앞의 경우와 마찬가지로 점차적으로 증가한다. 이를 통해 잔차넷은 앙상블 모델에서 관측될 수 있는 특성을 지닌다고 할 수 있다.

마지막 실험에서는 잔차넷이 상대적으로 얇은 층의 넷들의 앙상블 모델로 여겨질 수 있다는 사실을 보이고자 한다.



(a) (b) (c)

그림 9.27 (a) 잔차넷에서 표현 가능한 경로 길이의 분포는 이항분포로 표현될 수 있다. (b) 첫 번째 층에서 다른 길이의 경로를 지나면서 에러 경사가 나타나는지 살펴보았다. 경로에 놓여있는 모듈 수에 따라 대략적으로 지수적으로 감소함을 관찰하였다 (y축은 로그 척도). (c) 앞의 두 함수를 곱하여 특정 길이의 경로에서 얼만큼의 에러 경사가 기여하는지 관찰하였다. 20층 이상의 경로는 에러 경사가 충분히 기여하지 못하였다. (Veit et el., 2016)

그림 9.27 의 (a)와 (b)는 각각 이항분포로 나타낼 수 있는 잔차넷의 표현 가능한 경로의 수, 그리고 경로의 길이에 따른 에러 경사의 크기를 로그 크기로 나타내었다. (c)에서는 이 두 함수를 곱함으로써 도식화하였다. 잔차넷의 모델 구조 상에서 경로 길이는 25~30 이 가장 많지만 길이가 길수록 에러 경사의 기여가 지수적으로 감소하기 때문에 종합적으로는 5 에서 17 길이의 경로에서 가장 많은 기여가 있다고 추측할 수 있다. 확인하는 의미에서 실험에 사용된 54 층의 넷에서 23 개의 가장 효과적이라고 추측되는 층을 추출하여 효과적인 길이로 구성하여 실험하였을 때 5.96%, 전체 모델에서 6.10%의 에러를 보여 통계적으로 유의미한 차이를 관찰할 수 없었다. 따라서, 잔차넷은 상대적으로 얇은 층의 여러 넷으로 이루어진 암묵적인 앙상블 모델로 볼 수 있을 것이다.

딥 잔차 학습에 대한 연구가 공개된 지 일 년이 채 지나지 않아 주목할 만한 후속 연구가 다수 나왔다. Dropout 정규화 방법과 보다 큰 차원의 은닉층으로 구성된 잔차넷이 제안되기도 하였고(Singh et el., 2016), 잔차 연결선을 재귀적으로 연결하여 성능을 향상시킨 연구도 있었다(Zhang et el., 2016). 또, 기존의 잔차넷을 멀티모달로 확장한 멀티모달 잔차 학습 방법은 시각적 질의 응답 시스템을 구현하기 위해 적용되었다(Kim et el., 2016).

[참고문헌]

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., & Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems (pp. 2672-2680).*

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition In *Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Identity mappings in deep residual networks. In *arXiv preprint arXiv:1603.05027*, 2016

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation, 9(8), 1735-1780.*

https://en.wikipedia.org/wiki/Turing_machine

Kim, J. H., Lee, S. W., Kwak, D. H., Heo, M. O., Kim, J., Ha, J. W., & Zhang, B. T. (2016). Multimodal Residual Learning for Visual QA. In *Proceedings of Advances in neural information processing systems 29*, 2016.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Aitken, A., Tejani, A., ... & Shi, W. (2016). Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In *arXiv preprint arXiv:1609.04802*, 2016.

Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., & Efros, A. A. (2016). Context Encoders: Feature Learning by Inpainting. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.
Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations,* 2015.

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. (2016). Generative adversarial text to image synthesis. In *Proceedings of the 33nd International Conference on Machine Learning,* 2016.

Siegelmann, H. T., & Sontag, E. D. (1992, July). On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory (pp. 440-449).* ACM.

Singh, S., Hoiem, D., & Forsyth, D. (2016). Swapout: Learning an ensemble of deep architectures. In *Proceedings of Advances in neural information processing systems 29*, 2016.

Sukhbaatar, S., Weston, J., & Fergus, R. (2015). End-to-end memory networks.. In *Proceedings of Advances in Neural Information Processing Systems.* 2015.
Weston, J., Chopra, S., & Bordes, A. (2014). Memory networks. In *Proceedings of International Conference of Learning representations.* 2015.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems (pp. 3104-3112).*

Veit, A., Wilber, M., & Belongie, S. (2016). Residual Networks are Exponential Ensembles of Relatively Shallow Networks. In *Proceedings of Advances in neural information processing systems 29*, 2016.

Vinyals, O., & Le, Q. (2015). A neural conversational model. arXiv preprint arXiv:1506.05869.

Zhang, K., Sun, M., Han, T. X., Yuan, X., Guo, L., & Liu, T. (2016). Residual Networks of Residual Networks: Multilevel Residual Networks. In *arXiv Preprint arXiv1608.02908*, 2016.