TensorFlow 2.0으로 배우는 딥러닝 입문

순환신경망(RNN)

에이아이스쿨(AISchool) 대표
양진호 (솔라리스)
http://aischool.ai
http://solarisailab.com

강의 목표

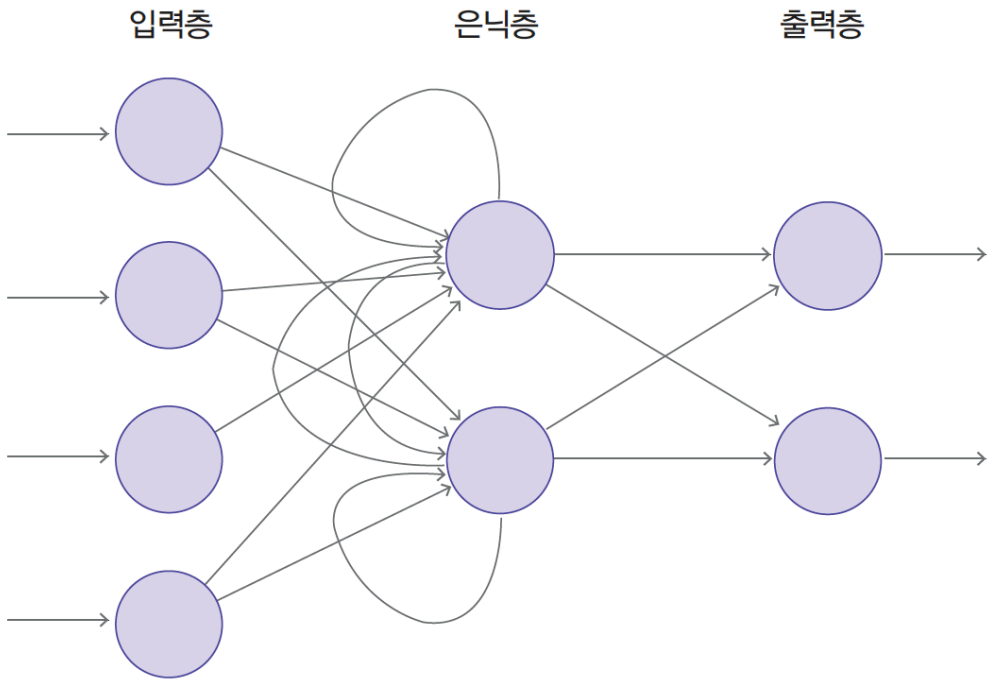
1. 순환신경망 (RNN)의 구조와 개념을 이해합니다.
2. Vanishing Gradient Problem, LSTM, GRU의 개념을 이해합니다.
3. 임베딩(Embedding)의 개념을 이해합니다.
4. TensorFlow 2.0을 이용해서 Char-RNN 구조를 구현해봅니다.

순환 신경망(RNN)

- CNN이 컴퓨터 비전Computer Vision 문제에 주로 사용되는 인공신경망 구조라면 이번 장에서 배울 **순환신경망**Recurrent Neural Networks(RNN)은 **자연어 처리**Natural Language Processing(NLP) 문제에 주로 사용되는 인공신경망 구조입니다.
- 좀 더 정확히 말하면, RNN은 시계열 데이터를 다루기에 최적화된 인공신경망입니다.
- **시계열 데이터**란 시간축을 중심으로 현재 시간의 데이터가 앞, 뒤 시간의 데이터와 연관 관계를 가지고 있는 데이터를 의미합니다. 예를 들어, 오늘의 주식 가격은 어제의 주식 가격과 연관이 있고, 내일의 주식 가격은 오늘의 주식 가격과 연관이 있습니다. 따라서 주식 가격은 시계열 데이터로 볼 수 있습니다.
- 주식 가격 이외에도 파형으로 표현되는 음성 데이터, 앞뒤 문맥을 가진 단어들의 집합으로 표현되는 자연어 데이터 등이 대표적인 시계열 데이터입니다.

순환 신경망(RNN)

- 이제 RNN 구조를 구체적으로 살펴봅시다. 아래 그림은 RNN의 구조를 나타냅니다.
- RNN은 기본적인 ANN 구조에서 이전 시간(t-1)의 은닉층의 출력값을 다음 시간(t)에 은닉층의 입력값으로 다시 집어넣는 경로가 추가된 형태입니다. 이 구조는 **“recurrent(순환되는)”**라는 단어에서 알 수 있듯이, 현재 시간 t의 결과가 다음 시간 t+1에 영향을 미치고, 이는 다시 다음 시간 t+2에 영향을 미치는 과정이 끊임없이 반복되는 인공신경망 구조입니다.



순환 신경망(RNN)

- 구체적으로 길이 T를 가진 입력 시퀀스 x를 I개의 인풋노드, H개의 히든노드, K개의 아웃풋노드를 가진 RNN에 입력하는 상황을 가정해봅시다. 이때 x_i^t는 시간 t 일때 i번째 입력데이터, a_h^t는 각각 시간 t일때 히든 레이어의 출력값을 나타냅니다. $w_{h\prime h} b_{h\prime}^{t-1}$는 이전 시간(t-1)의 히든 유닛의 활성값과 히든 유닛에서 히든 유닛으로 되돌아오는 가중치의 곱을 나타냅니다.

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h\prime=1}^{H} w_{h\prime h} b_{h\prime}^{t-1}$$

- 최종적으로 a_h^t 에 Sigmoid나 ReLU 같은 활성 함수 σ_h 를 씌워 **히든 유닛의 활성값 b_h^t** 를 계산합니다.

$$b_h^t = \sigma_h(a_h^t)$$

- 전체 시퀀스의 히든 유닛 활성값은 시간 t를 1부터 끝까지 증가시키면서 위의 수식을 계산함으로써 얻을 수 있습니다. 시퀀스의 처음 시작(t=0)일 때는 이전 시간의 히든 유닛의 활성값이 없기 때문에 임의로 지정해주어야 합니다.

- 보통은 **초기 활성값인 b_h^0 을 0으로 초기화**합니다.

순환 신경망(RNN)

- **출력층**에서는 일반적인 ANN과 같은 방식으로 은닉층으로터 주어지는 입력값과 가중치를 곱하고 합해서 구한 출력값과 a_{out}^{t} 에 활성 함수를 적용해서 출력 유닛의 최종 출력값 b_{out}^{t} 을 계산합니다.

$$a_k^t = \sum_{h=1}^{H} w_{hk} x_h^t$$

$$b_{out}^t = \sigma_{out}(a_{out}^t)$$

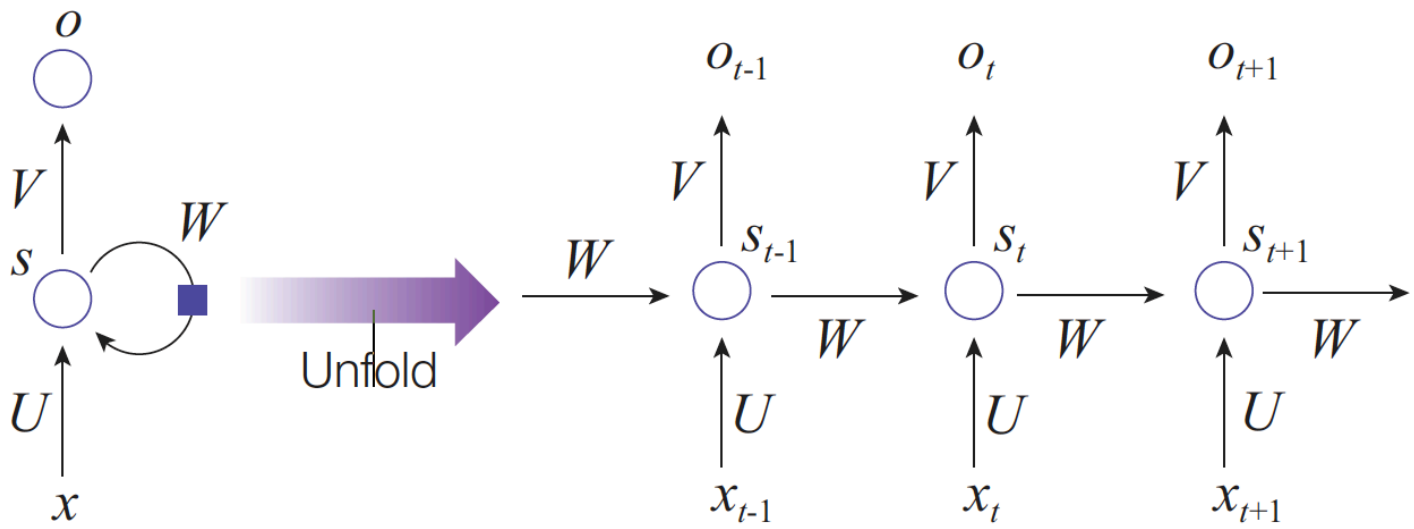
- 이런 구조를 통해 얻을 수 있는 **장점은 이전 상태에 대한 정보를 일종의 메모리Memory 형태로 저장할 수 있다는 점**입니다. 이는 앞에서 얻은 정보가 다음에 얻은 데이터와 연관 관계를 가지는 시계열 데이터를 다룰 때 매우 강력한 효과를 발휘합니다.

- 구체적인 예로 인간의 언어Natural Language는 앞뒤 문맥Context을 가지고 있기 때문에 RNN을 적용하기에 매우 적합합니다.

- 예를 들어 “푸른 하늘에 OO이 떠있다.”라는 문장에서 OO에 들어갈 단어를 예측하고자 한다면, 우리는 앞뒤 정보인 “푸른”, “하늘”, “떠있다.”라는 단어를 통해 OO에 들어갈 단어가 “구름”이라는 것을 쉽게 예측할 수 있습니다.

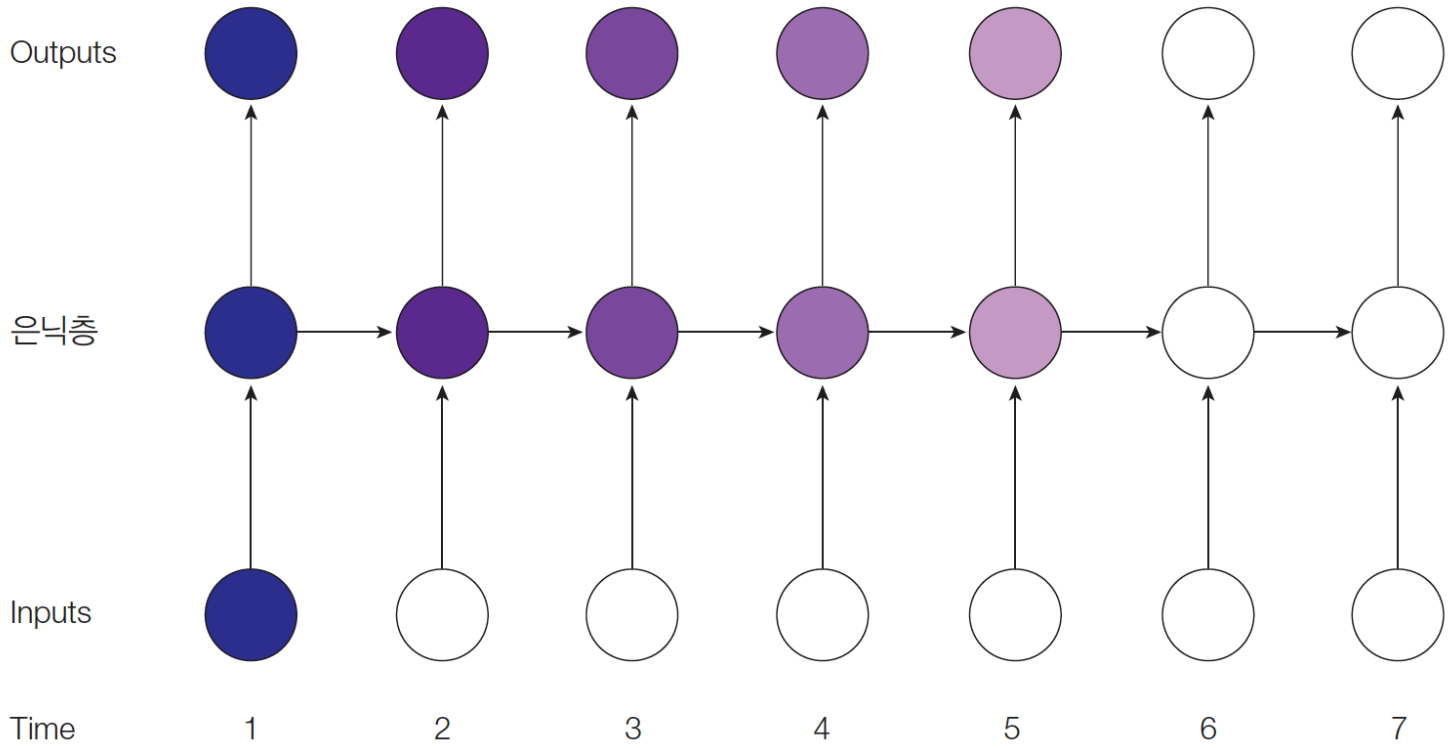
순환 신경망(RNN)

- RNN을 다른 관점으로 바라보면, 시간축에 따라 인공신경망을 펼친Unfold 형태로 생각할 수 도 있습니다.
- 예를 들어, 5개의 단어로 이루어진 문장을 RNN의 인풋으로 사용한다면, 순환 연결이 없는 인공신경망을 5층으로 쌓은 것으로 바라볼 수 있습니다.
- 아래 그림은 unfold 형태의 RNN을 나타냅니다.



LSTM(장단기 기억 네트워크)와 경사도 사라짐 문제

- RNN은 시계열 데이터를 다루기에 적합하지만, **경사도 사라짐 문제**Vanishing Gradient Problem가 있습니다. 그림은 경사도 사라짐 문제의 예를 보여줍니다.

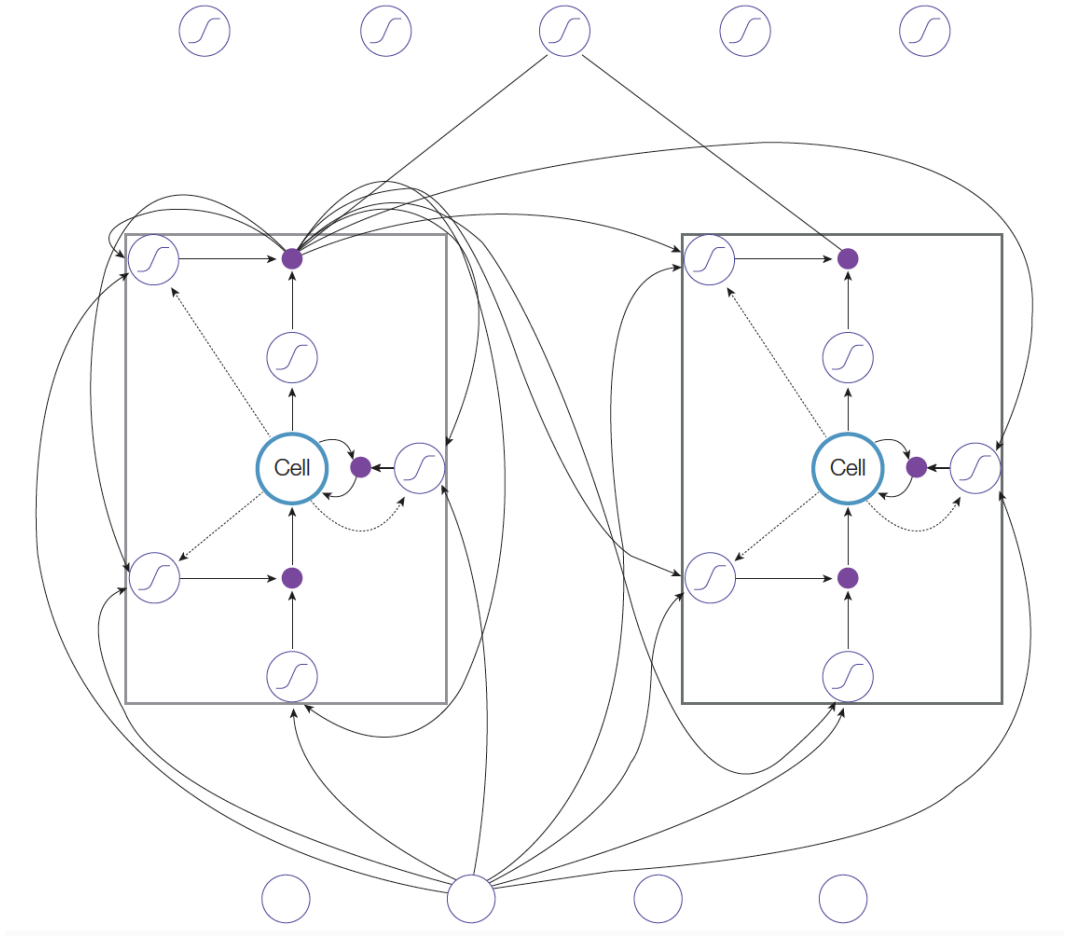


LSTM(장단기 기억 네트워크)와 경사도 사라짐 문제

- 시간 1에서 입력받은 데이터는 시간 1에서 RNN의 파라미터를 업데이트하거나 예측을 진행하는데 강한 영향력(=큰 경사도Gradient)을 끼칩니다. 하지만 시간 2, 3, … 계속해서 새로운 데이터가 들어옴으로써 새로 들어온 데이터의 영향력에 덮어 쓰여서, 시간 1에서 입력 받은 데이터의 영향력(=작은 경사도)은 조금씩 사라지다가 시간 7 쯤에서 영향력이 완전히 사라지는 현상을 관찰할 수 있습니다.
- 이런 현상을 **경사도 사라짐 문제**Vanishing Gradient Problem라고 부릅니다.
- 이로 인한 문제점은 **RNN이 장기기억력을 가지지 못한다는 점**입니다. 다시 말하면 RNN은 경사도 사라짐 문제로 인해서 현재 시간에서 가까운 시간의 데이터만 고려해서 예측을 진행하게 되고, 이는 현재 시간보다 오래 전의 데이터를 고려해서 예측해야하는 상황에서 RNN의 성능이 감소하는 결과를 낳습니다.
- **LSTM(장/단기 기억 네트워크)**Long-Short Term Memory Networks는 이런 경사도 사라짐 문제를 해결하기 위해서 제안된 발전된 RNN 구조입니다.
- LSTM은 은닉층의 각각의 노드를 **인풋 게이트**Input Gate, **포겟 게이트**Forget Gate, **아웃풋 게이트**Output Gate로 구성된 **메모리 블럭**Memory Block이라는 조금은 복잡한 구조로 대체합니다.
- 인풋 게이트, 포겟 게이트, 아웃풋 게이트를 이용해서 경사도 사라짐 문제를 완화할 수 있습니다.

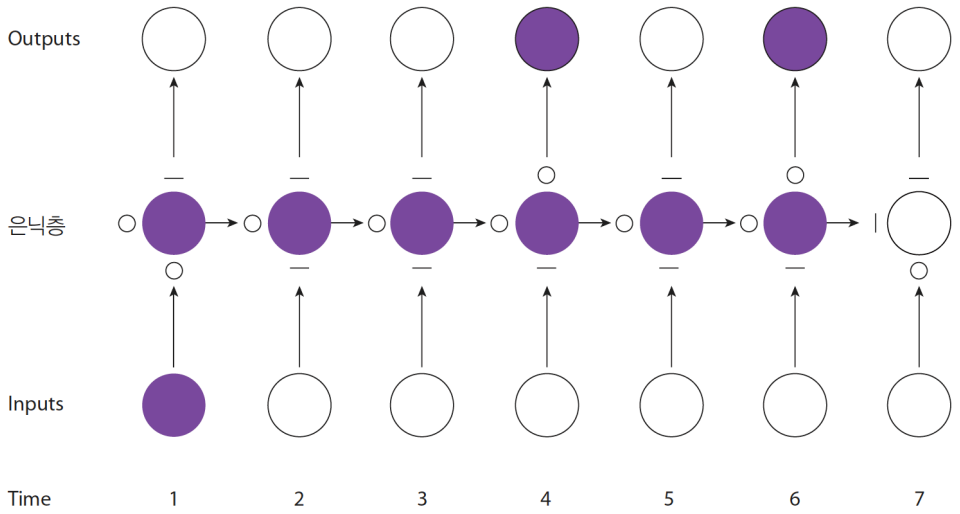
LSTM(장단기 기억 네트워크)와 경사도 사라짐 문제

- 아래 그림는 LSTM의 전체 구조를 보여줍니다.



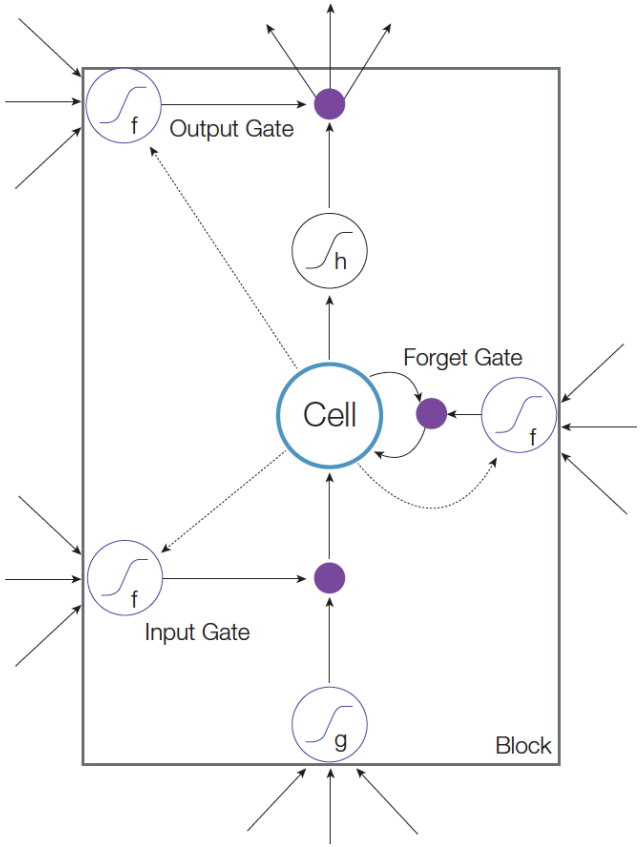
LSTM(장단기 기억 네트워크)와 경사도 사라짐 문제

- 그림은 LSTM이 메모리 블럭을 이용해서 어떻게 경사도 사라짐 문제를 해결하는지 보여줍니다.
- LSTM은 인풋 게이트, 포겟 게이트, 아웃풋 게이트를 열고 닫으면서 시간 1의 영향력을 오랫동안 가져갈 수 있습니다.
- 그림의 은닉층의 아래쪽은 인풋 게이트, 은닉층의 왼쪽은 포겟 게이트, 은닉층의 위쪽은 아웃풋 게이트를 나타냅니다. 또한 **O는 게이트가 열린 상태, ㅡ은 게이트가 닫힌 상태**를 나타냅니다.
- 그림에서 나타내는 예제의 경우, 시간 1에서 인풋 게이트를 열어서 시간 1의 인풋 데이터의 영향력을 가져가고, 시간 6까지 포겟 게이트를 열어서 시간 1의 영향력을 계속해서 다음 시간으로 가져갑니다.
- 하지만 시간 2에서부터 6까지는 인풋 게이트를 닫아서 시간 2에서부터 6까지의 인풋 데이터 영향력을 없애버립니다. 이를 통해서 시간 1의 인풋 데이터 영향력이 새로운 인풋 데이터에 의해서 덮어씌워지는 문제를 해결할 수 있습니다. 또한 시간 1, 2, 3, 5에서는 아웃풋 게이트를 닫고 시간 4, 6에서만 아웃풋 게이트를 열어서 RNN의 출력 결과를 시간 4, 6에서만 방출할 수 있습니다.



LSTM(장단기 기억 네트워크)와 경사도 사라짐 문제

- 이제 인풋 게이트, 포겟 게이트, 아웃풋 게이트에서 일어나는 연산들을 더 구체적으로 살펴봅시다.
- 그림은 하나의 메모리 블럭을 나타냅니다.



LSTM(장단기 기억 네트워크)와 경사도 사라짐 문제

- 먼저 **인풋게이트**에서 일어나는 연산을 살펴봅시다. 인풋 게이트에서는 현재 시간 t의 입력 데이터 x_i^t에 인풋게이트로 이어지는 가중치 w_{iI}를 곱해서 더한 값과 이전 시간 t-1의 메모리 블럭 출력값 b_h^{t-1}과 출력값 s_c^{t-1}에 인풋게이트로 이어지는 가중치 w_{hI}, w_{cI}를 곱한 값을 더해서 인풋게이트의 출력값 a_I^t 를 구하게 됩니다. 이 값에 최종적으로 활성화 함수 f를 적용해서 인풋 게이트의 출력값 b_I^t 를 계산하게 됩니다.

$$a_I^t = \sum_{i=1}^{I} w_{iI} x_i^t + \sum_{h=1}^{H} w_{hI} b_h^{t-1} + \sum_{c=1}^{C} w_{cI} s_c^{t-1}$$
$$b_I^t = f(a_I^t)$$

- 다음으로 **포겟 게이트**에서 일어나는 연산을 살펴봅시다. 포겟 게이트는 인풋게이트와 똑같은 형태로 단지 곱해지는 가중치가 포겟게이트로 이어지는 가중치로 바뀐 것뿐입니다. 좀더 자세하게 설명하면, 포겟 게이트에서는 현재 시간 t의 입력 데이터 x_i^t에 포겟게이트로 이어지는 가중치 w_{iF}를 곱해서 더한 값과 이전 시간 t-1의 메모리 블럭 출력값 b_h^{t-1}과 출력값 s_c^{t-1}에 포겟게이트로 이어지는 가중치 w_{hF}, w_{cF}를 곱한 값을 더해서 포겟게이트의 출력값 a_F^t 를 구하게 됩니다. 이 값에 최종적으로 활성화 함수 f를 적용해서 포겟게이트의 출력값 b_F^t 를 계산하게 됩니다.

$$a_F^t = \sum_{i=1}^{I} w_{iF} x_i^t + \sum_{h=1}^{H} w_{hF} b_h^{t-1} + \sum_{c=1}^{C} w_{cF} s_c^{t-1}$$
$$b_F^t = f(a_F^t)$$

LSTM(장단기 기억 네트워크)와 경사도 사라짐 문제

- 다음으로 블럭 인풋 게이트Block Input Gate에서 일어나는 연산을 살펴봅시다.
- 블럭 인풋 게이트는 그림 1-1의 메모리 블럭의 아래쪽 중앙에 있는 게이트로 정식 명칭은 없지만 메모리 블럭의 입력값에 활성함수를 씌워주는 게이트입니다. 이 책에서는 편의상 블럭 인풋 게이트라고 명명하겠습니다.
- 블럭 인풋게이트에서는 현재 시간 t의 입력 데이터 x_i^t에 블럭 인풋게이트로 이어지는 가중치 w_{ic}를 곱해서 더한 값과 이전 시간 t-1의 메모리 블럭 출력값 b_h^{t-1}과 블럭 인풋게이트로 이어지는 가중치 w_{hc}를 곱한 값을 더해서 블럭 인풋게이트의 출력값 a_c^t 를 구하게 됩니다. 이 값에 최종적으로 활성화 함수 g를 적용해서 블럭 인풋게이트의 출력값 $g(a_c^t)$ 를 계산하게 됩니다.

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1}$$

$$g(a_c^t) = g(a_c^t)$$

LSTM(장단기 기억 네트워크)와 경사도 사라짐 문제

- 이제 메모리 블럭 중간에 있는 **셀에서 일어나는 연산**을 살펴봅시다. 현재 시간의 셀 출력값 s_c^t는 이전 시간 t-1의 셀의 출력값 s_c^{t-1}과 현재시간 t의 포겟 게이트의 출력값 b_F^t을 곱한 값에 현재시간 t의 블럭 인풋게이트의 출력값 $g(a_c^t)$에 인풋 게이트의 출력값 b_I^t을 곱한 값을 더해서 계산합니다.

$$s_c^t = b_F^t s_c^{t-1} + b_I^t \, g(a_c^t)$$

- 이렇게 **셀에서 일어나는 연산이 인풋 게이트와 포겟 게이트를 열고 닫아서 현재 시간과 이전 시간의 데이터의 영향력을 가져갈지 혹은 잊어버릴지Forget를 결정**하게 됩니다.

- 셀에서 일어나는 연산을 좀더 직관적으로 생각해보기 위해서 포겟게이트의 출력값 b_F^t와 인풋게이트의 출력값 b_I^t를 α,β로 생각해봅시다. 그러면 셀에서 일어나는 연산은 다음과 같이 표현할 수 있습니다.

$$s_c^t = \alpha s_c^{t-1} + \beta g(a_c^t)$$

LSTM(장단기 기억 네트워크)와 경사도 사라짐 문제

- 이때 α가 0이면 이전 시간 t-1의 셀 출력값 s_c^{t-1}의 영향력이 0이 되어서 현재 시간 t의 셀 출력값은 아래와 같이 오직 블럭 인풋게이트의 영향력만을 고려하는 상태가 됩니다.

$$s_c^t = \beta g(a_c^t)$$

- 이는 이전 그림에서 인풋 게이트를 열고 포겟 게이트는 닫는 상태로 생각할 수 있습니다. 이에 반해 β가 0이라면 현재 시간 t의 셀 출력값이 아래와 같이 오직 이전 시간의 셀 출력값만을 고려하는 상태가 됩니다.

$$s_c^t = \alpha s_c^{t-1}$$

- 이는 이전 그림의 인풋 게이트를 닫고 포겟 게이트를 연 상태로 볼 수 있습니다. 즉 게이트의 출력값이 0이라면 게이트를 닫고 출력값이 1이면 게이트를 여는 행위로 볼 수 있습니다. 따라서 **인풋게이트의 출력값은 현재 시간 t에서 받은 인풋 데이터의 영향력을 반영할지 혹은 반영하지 않을지를 결정**하고, **포겟게이트의 출력값은 이전 시간에 있던 인풋데이터의 영향력을 반영할지 혹은 반영하지 않을지를 결정**합니다.

- 실제 상황에서는 0 혹은 1의 값으로 정확히 게이트를 열고 닫지 않고 0~1 사이의 적절한 값을 갖게 됩니다. 이 모든 과정은 오류역전파를 이용한 최적화를 이용해서 데이터를 표현하기에 가장 적합한 형태로 자동으로 최적화됩니다.

LSTM(장단기 기억 네트워크)와 경사도 사라짐 문제

- 마지막으로 아웃풋 게이트에서 일어나는 연산을 살펴봅시다.
- 아웃풋 게이트에서는 현재 시간 t의 입력 데이터 x_i^t에 아웃풋게이트로 이어지는 가중치 w_{iO}를 곱해서 더한 값과 이전 시간 t-1의 메모리 블럭 출력값 b_h^{t-1}과 현재시간 t의 셀 출력값 s_c^t에 아웃풋 게이트로 이어지는 가중치 w_{hO}, w_{cO}를 곱한 값을 더해서 아웃풋 게이트의 출력값 a_O^t 를 구하게 됩니다. 이 값에 최종적으로 활성화 함수 f를 적용해서 아웃풋 게이트의 출력값 b_O^t 를 계산하게 됩니다.

$$a_O^t = \sum_{i=1}^{I} w_{iO} x_i^t + \sum_{h=1}^{H} w_{hO} b_h^{t-1} + \sum_{c=1}^{C} w_{cO} s_c^t$$

$$b_O^t = f(a_O^t)$$

LSTM(장단기 기억 네트워크)와 경사도 사라짐 문제

- 최종적으로 아웃풋 게이트의 출력값 b_o^t에 현재 시간 t의 셀 상태값에 활성함수를 씌운 값 $h(s_c^t)$을 곱해서 현재 시간 t의 메모리 블럭에서 방출하는 값 b_c^t를 계산할 수 있습니다.

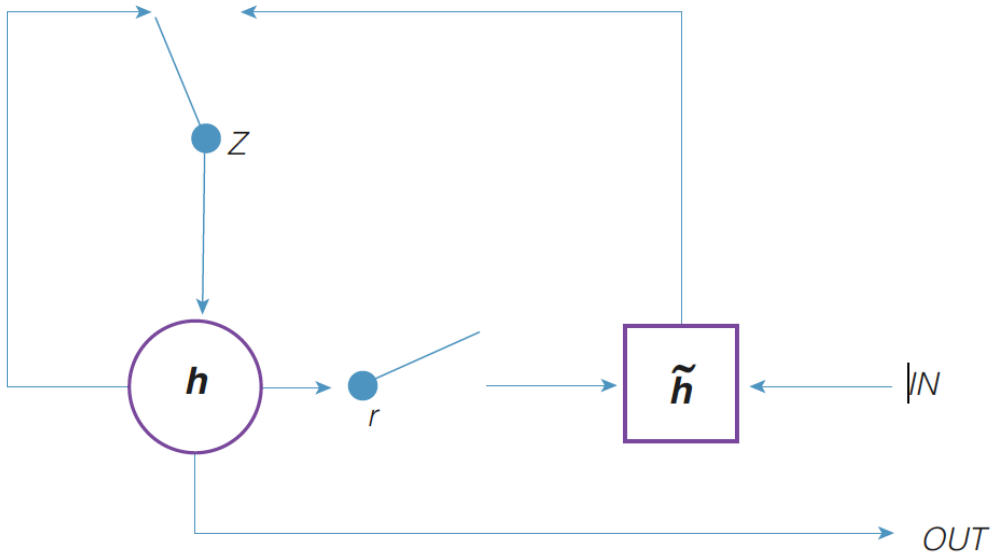
$$b_c^t = b_o^t h(s_c^t)$$

- 여기서도 마찬가지로 아웃풋 게이트의 출력값이 0이라면 메모리 블럭이 방출하는 값이 0이 되는 아웃풋 게이트가 닫힌 상태, 아웃풋 게이트의 값이 1이라면 상태값을 전부 외부로 방출하는 아웃풋 게이트가 열린 상태로 간주할 수 있습니다.

- 지금까지 LSTM이 구체적으로 어떻게 경사도 사라짐 문제를 해결하는지 살펴보았습니다. 시계열 데이터에 RNN을 사용하는 경우, 장기 기억이 필요한 경우가 대부분이므로 일반적으로 RNN을 사용한 경우보다 LSTM을 사용하는 경우가 더욱 좋은 성능을 보여줍니다. 하지만 LSTM은 RNN보다 더 많은 연산이 필요하므로 더많은 학습시간과 컴퓨팅 파워가 필요합니다

GRU(Gate Recurrent Unit)

- **GRU**$_{\text{Gate Recurrent Unit}}$의 LSTM의 간략화된 버전입니다. GRU는 LSTM의 3개의 게이트를 2개의 게이트로 축소하였습니다. 구체적으로 GRU는 **리셋 게이트**$_{\text{Reset Gate}}$ **r**과 **업데이트 게이트**$_{\text{Update Gate}}$ **z**를 가지고 있습니다.
- 그림은 GRU의 메모리 블럭을 나타냅니다.



GRU(Gate Recurrent Unit)

- 각각의 게이트에서 일어나는 연산을 구체적으로 살펴봅시다.
- **리셋 게이트의 출력값 z**는 현재 시간 t의 인풋데이터를 x_t에 리셋 게이트로 이어지는 가중치 U^z를 곱한 값과 이전 시간 t-1의 메모리 블럭의 출력값 s_{t-1}에 리셋 게이트로 이어지는 가중치 W^z를 곱해서 더한값에 활성함수 σ를 씌워서 계산합니다.

$$z = \sigma(x_t U^z + s_{t-1} W^z)$$

- **업데이트 게이트는 r**은 현재 시간 t의 인풋데이터를 x_t에 업데이트 게이트로 이어지는 가중치 U^r를 곱한 값과 이전 시간 t-1의 메모리 블럭의 출력값 s_{t-1}에 리셋 게이트로 이어지는 가중치 W^r를 곱해서 더한값에 활성함수 σ를 씌워서 계산합니다.

$$r = \sigma(x_t U^r + s_{t-1} W^r)$$

GRU(Gate Recurrent Unit)

- **메모리 블럭의 내부 셀 출력값 h**는 현재 시간 t의 인풋데이터를 x_t에 내부 셀로 이어지는 가중치 U^h를 곱한 값과 이전 시간 t-1의 메모리 블럭의 출력값 s_{t-1}에 리셋 게이트의 출력값 r을 곱하고 내부 셀로 이어지는 가중치 W^h를 곱해서 더한값에 활성함수 tanh를 씌워서 계산합니다.

$$h = \tanh(x_t U^h + (s_{t-1} * r)W^h)$$

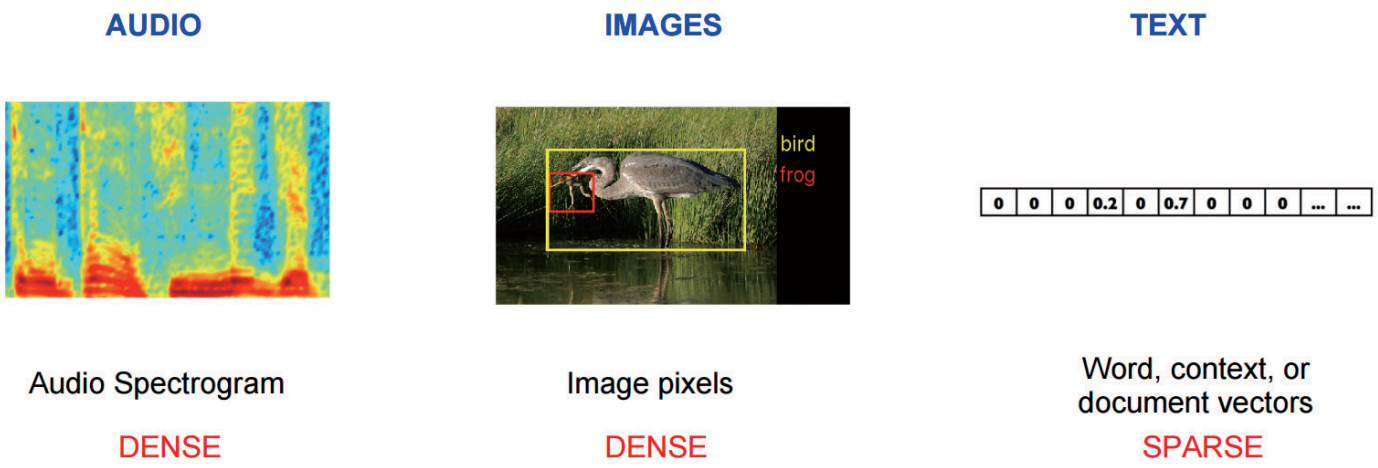
- 현재 시간 t의 메모리 블럭의 출력값 s_t는 메모리 블럭의 내부 셀 출력값 h에 (1-z)를 곱한 값에 리셋 게이트이 출력값 z와 이전 시간 t-1의 메모리 블럭의 출력값 s_{t-1}을 곱해서 계산합니다.

$$s_t = (1 - z) * h + z * s_{t-1}$$

- 정리하면 GRU는 LSTM에서 출력 게이트를 제거하고, 인풋 게이트와 포겟 게이트의 역할을 업데이트 게이트와 리셋 게이트가 나눠 가져간 형태로 볼 수 있습니다. 보통 LSTM이 조금 더 강력한 성능을 보여주는 것으로 알려져 있지만 GRU도 LSTM과 비슷한 성능을 보여주면서 연산량이 감소되었기 때문에 컴퓨팅 환경이 좋지 못한 경우 LSTM 대신 GRU를 사용하는 것도 좋은 선택이 될 수 있습니다.

임베딩Embedding의 개념

- **임베딩Embedding**은 머신러닝 알고리즘을 사용할 때, 그 중에서도 특히 자연어 처리 문제를 다룰 때 널리 사용되는 기법입니다.
- 머신러닝 알고리즘을 사용할 때 데이터를 표현하는 일반적인 방법은 One-hot Encoding입니다. 하지만 아래 그림에서 볼 수 있듯이 One-hot Encoding은 데이터의 표현 형태가 Sparse하다는 문제점이 있습니다.
- 예를 들어 10,000개의 단어사전에 있는 단어 하나를 One-hot Encoding으로 표현하면 10,000×1의 행렬에서 1개의 행에만 1이라는 값이 있고, 나머지 9999개의 행에는 0이라는 의미없는 값이 들어가 있을 것입니다. 결과적으로 표현에서 잉여 부분이 많아집니다. 또한 One-hot Encoding은 유사한 의미를 단어를 가진 단어간의 연관성도 표현할 수 없습니다.

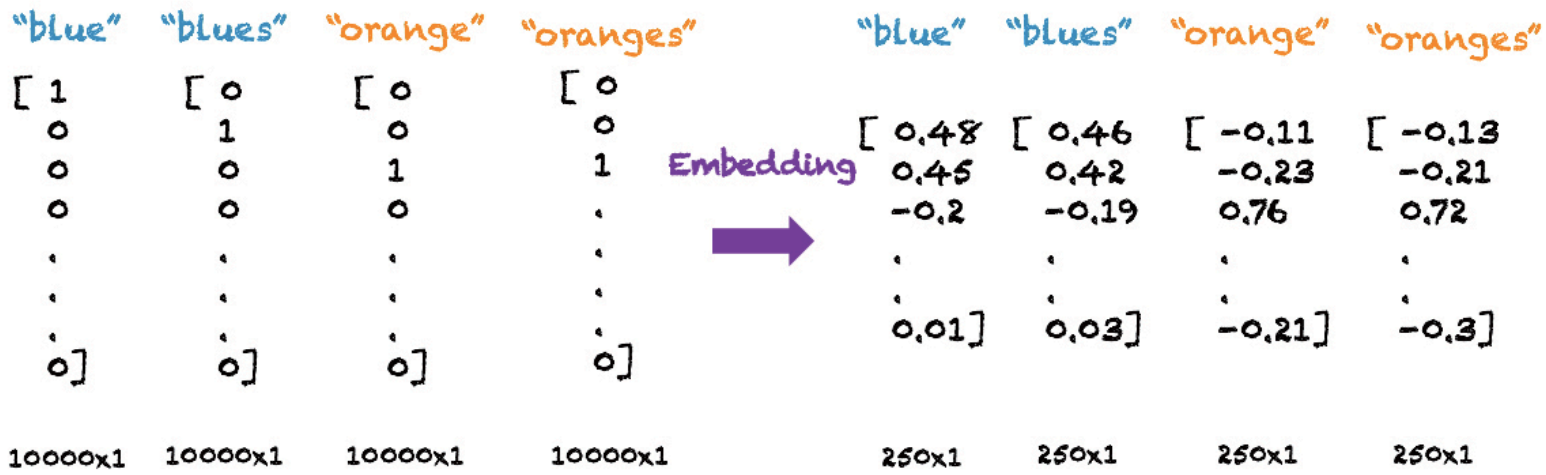


임베딩Embedding의 개념

- **임베딩Embedding**은 이러한 문제점을 해결하기 위해서 Sparse한 One-hot Encoding의 데이터 표현을 Dense한 표현형태로 변환하는 기법입니다. 이를 위해서 원본 데이터에 Dense한 임베딩 행렬Embedding Matrix을 곱해서 데이터의 표현형태를 아래 수식처럼 변환합니다.

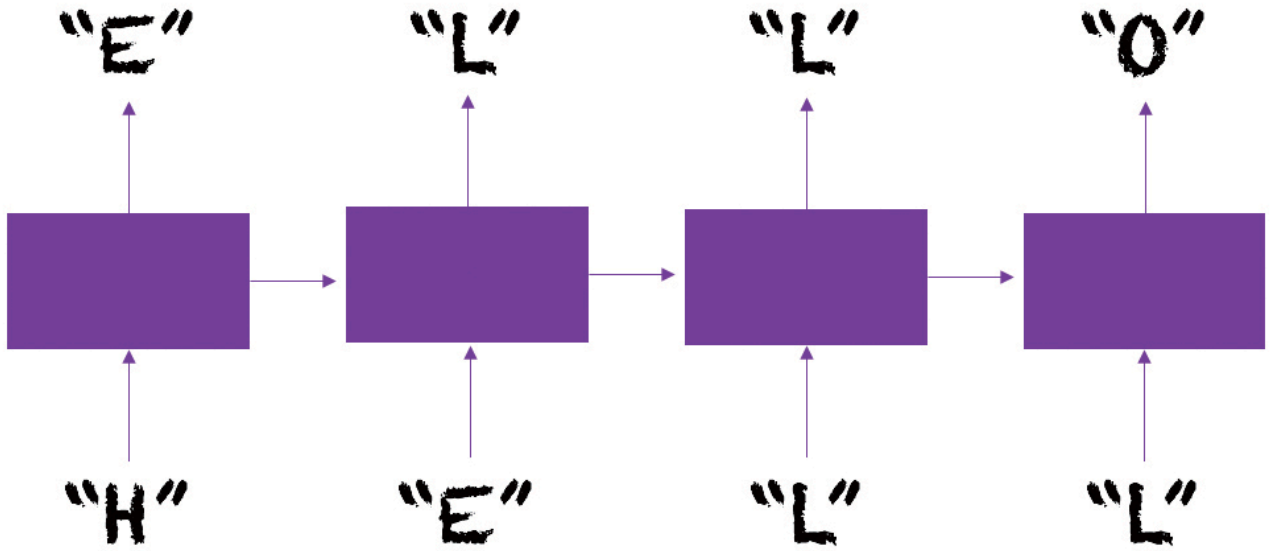
$$x_{embedding} = W_{embedding} x_{one-hot}$$

- 아래 그림은 10,000개의 단어사전을 One-hot Encoding으로 표현한 데이터에 10,000×250 크기의 임베딩 행렬을 곱해서 임베딩을 수행한 예시를 보여줍니다.



Char-RNN의 개념

- Char-RNN은 RNN을 처음 배울 때 가장 많이 사용되는 예제 중 하나로서 하나의 글자Character를 RNN의 입력값으로 받고, RNN은 다음에 올 글자를 예측하는 문제입니다.
- 이를 위해서 RNN의 **타겟 데이터를 인풋 문장에서 한 글자씩 뒤로 민 형태**로 구성하면 됩니다. 예를 들어서 “HELLO”라는 문장을 학습시키고 싶을 경우, RNN의 (인풋 데이터, 타겟 데이터) 쌍을 (H, E), (E,L), (L, L), (L,O)로 구성합니다.
- 그림은 Char-RNN을 이용해서 HELLO라는 문장을 학습시키는 예제를 보여줍니다.

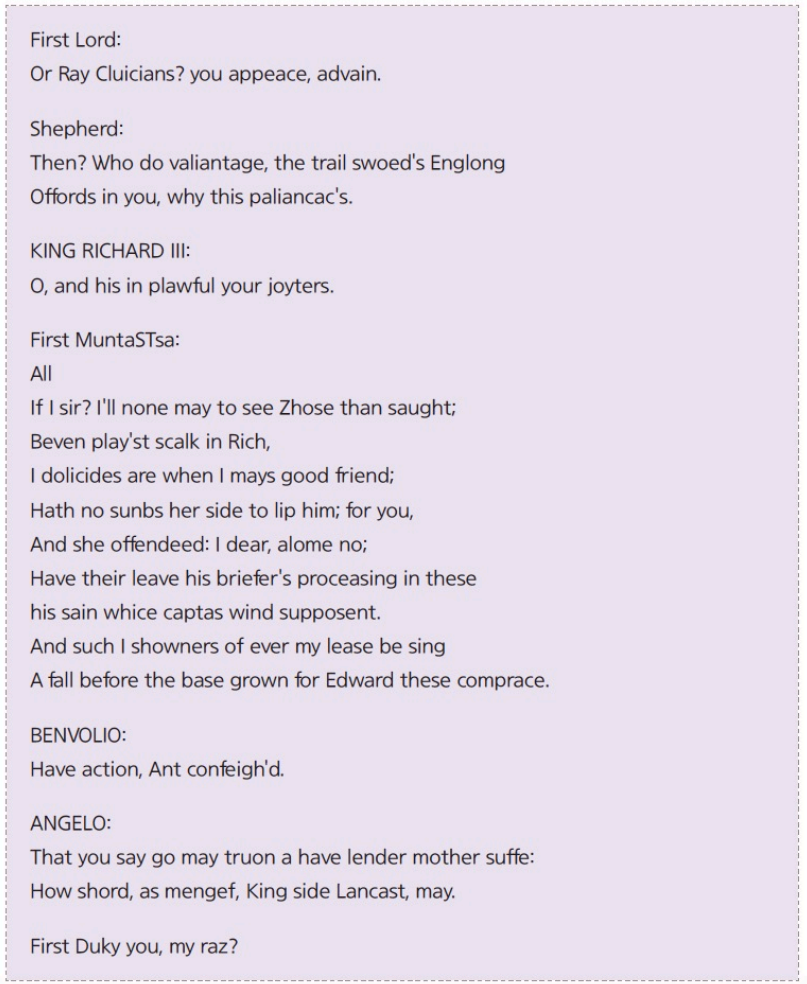


Char-RNN의 개념

- 이때 Char-RNN의 출력값 형태는 학습에 사용하는 전체 문자(단어$_{Vocabulary}$) 집합에 대한 소프트맥스 출력값이 됩니다. 따라서 영어 문자로만 구성된 데이터셋일 경우, 전체 문자 집합은 알파벳 글자 개수인 26이 될 것입니다.
- 즉, Char-RNN의 출력값은 다음에 올 26개의 알파벳 문자에 대한 확신의 정도를 나타내는 26×1 크기의 행렬이 될 것입니다.
- 그 중에서 argmax로 가장 확률이 높은 글자를 다음에 올 글자로 확정하고 그 글자를 이용해서 또 다음에 올 글자를 예측하는 과정을 반복합니다.

Char-RNN의 개념

- 셰익스피어의 희곡 <리처드 3세>로 Char-RNN을 학습하고 샘플링한 결과 중 일부 발췌하면 그림과 같습니다.



First Lord:
Or Ray Cluicians? you appeace, advain.

Shepherd:
Then? Who do valiantage, the trail swoed's Englong
Offords in you, why this paliancac's.

KING RICHARD III:
O, and his in plawful your joyters.

First MuntaSTsa:
All
If I sir? I'll none may to see Zhose than saught;
Beven play'st scalk in Rich,
I dolicides are when I mays good friend;
Hath no sunbs her side to lip him; for you,
And she offendeed: I dear, alome no;
Have their leave his briefer's proceasing in these
his sain whice captas wind supposent.
And such I showners of ever my lease be sing
A fall before the base grown for Edward these comprace.

BENVOLIO:
Have action, Ant confeigh'd.

ANGELO:
That you say go may truon a have lender mother suffe:
How shord, as mengef, King side Lancast, may.

First Duky you, my raz?

Char-RNN의 개념

- 한눈에 봐도 그럴듯한 텍스트를 생성해낸 모습을 볼 수 있습니다. 희곡은 Char-RNN의 학습 데이터로 널리 사용되는데 RNN이 학습하는 것은 글자 배열의 패턴이기 때문입니다.
- 희곡 같은 경우 문장의 첫 부분에 “등장인물:” 형태의 고정된 패턴이 반복해서 등장하기 때문에 Char-RNN의 학습에 적합합니다. 이전 그림 의 <리처드 3세> 샘플링 결과를 보면 <리처드 3세>의 실제 등장인물인 First Lord, Shepherd, KING RICHARD III, BENVOLIO, ANGELO 등의 이름을 정확히 생성해낸 모습을 볼 수 있습니다. 하지만 First MuntaSTsa와 같은 이상한 이름을 생성한 경우도 있습니다. 또한 생성한 영어 대사의 경우, 의미론적으로는 말이 안되지만 구조적으로는 그럴듯한 모습을 띄는 것을 알 수 있습니다.
- 이제 다른 학습 데이터로 재밌는 실험을 하나 더 해봅시다. 프로그래밍 언어 또한 언어의 문법에 따라 일정한 패턴으로 코드가 작성되기 때문에 Char-RNN의 학습에 적합합니다.
- 다음 그림은 C언어로 작성된 리눅스 소스 코드를 이용해서 Char-RNN을 학습시킨 후 샘플링한 결과입니다.

Char-RNN의 개념

- 희곡의 경우와 마찬가지로 의미로는 말이 안되지만 구조적으로는 그럴 듯한 모습을 하는 것을 알 수 있습니다.

```
void graph_index(struct hrtime_hardetr, bolaction, struct hectx *tsk)
{
  struct plog_batch;
  get_rrgbum_shifock_remove_elaper(L)) {
  dr.name_graph(return, tsk);
    if (index = &bay->count - (smap, hardid_usigned && tr->sched_name && ~SUPTION_
BT(032))
  return 1;

  raw_spin_unlock(dstat_val)))
  return -EINVAL;
  i+++)
       return -EINVA;
}

static inline bufferpoid(tp, char *timer, struct rq *rq)
{ }
#else
  return rq->ret_module_aps(p, i) gore, &path) &" factive it rcu\bled tsk0.
  *
    * Arce a dis:
  return error = 1;
  retuct task_next;

  if (hadd_piocup.ext, sizeof(queued);
    reSult = &context->trace_hung_twueued entry->graph_find_rmatf("Tode, infags
or */
  return prink;

  timekeeping_write(, p2))
    return;
  }
```

TensorFlow 2.0을 이용한 Char-RNN 구현

- 이제 Char-RNN의 개념을 이해했으니, 텐서플로 라이브러리를 이용해서 Char-RNN을 구현해봅시다.
- https://github.com/solaris33/deep-learning-tensorflow-book-code/blob/master/Ch08-RNN/Char-RNN/train_and_sampling_v2_keras.py

Chapter 7 - Convolutional Neural Networks(CNN)

- CNN을 이용한 MNIST 숫자 분류기 구현 (Code) (TF v2 Code) (TF v2 Keras Code)
- CNN을 이용한 CIFAR-10 분류기 구현 (Code) (TF v2 Code) (TF v2 Keras Code)
- tf.train.Saver API를 이용해서 모델과 파라미터를 저장(Save)하고 불러오기(Restore) (Code) (TF v2 Code) (TF v2 Keras Code)

Chapter 8 - Recurrent Neural Networks(RNN)

- tf.nn.embedding_lookup을 이용한 Embedding 예제 (Code)
- Gradient Clipping 예제 (Code) (TF v2 Keras Code)
- Char-RNN을 이용한 텍스트 생성 (Code) (TF v2 Keras Code)

Argmax 샘플링

hand,
That shall be the seem the seem the seement thee
The seadent the seem the seem the seement thee,
The shall be the seem the seem the seement thee
The seadent the seem the seem the seement thee,
The shall be the seem the seem the seement thee
The seadent the seem the seem the seement thee,
The shall be the seem the seem the seement thee
The seadent the seem the seem the seement thee,
The shall be the seem the seem the seement thee
The seadent the seem the seem the seement thee,
The shall be the seem the seem the seement thee
The seadent the seem the seem the seement thee,
The shall be the seem the seem the seement thee
The seadent the seem the seem the seement thee,
The shall be the seem the seem the seement thee
The seadent the seem the seem the seement thee,
The shall be the seem the seem the seement thee
The seadent the seem the seem the seement thee,
The shall be the seem the seem the seement thee
The seadent the seem the seem the seement thee,
The shall be the seem the seem th

Categorical Distribution 샘플링

lay upon as
Ay! what away your inventle ir is to desPips
True these assle to, which the iserate, come tell.

CORIOLANUS:
He she mouth fool–desideden to good since
For this absed upon, and kiss inst my bloode?
The way! And heffess of way; and them the jest:
If our seuzen bick the sir. Then with put him her.
Wone, the masters to his places is, neess against the bears any.
Lord, whence parden to masters show of moversalings,
And fixtings; I suvrace me, he sword duty with your daughter her this.

FLORIZEL:
So fall, and the mayhors, prinhest bettion.

BIRANDA:
The commass!
O, misician sulpt his babise hearthers,
That Rome you unmerlameous him dot himself,
For their loveness, conses, is and this good forth,
His officed denament, my lord.
Thank his have Lancys.

Second Sicinlier 'form thee place: but himbhire,
I rachence I will gravents Romeo the more sile
That strethe home?

First LCozizen::
She scond to sight Paging seema: him, as the forcenence
That live speak the brother hoarn you,
Monday

Thank you!