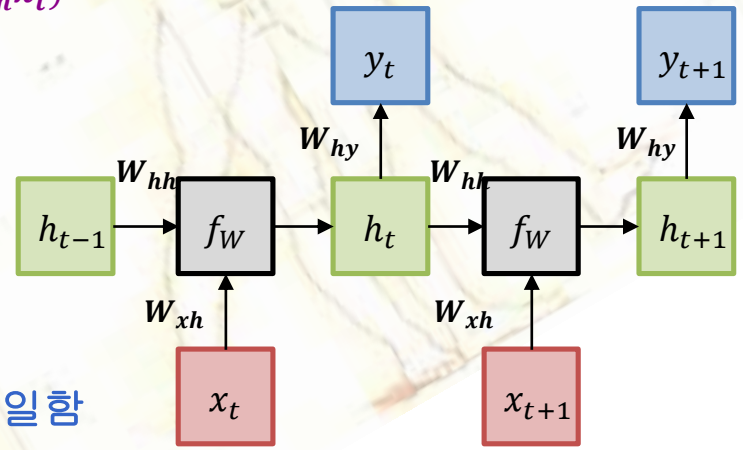
Recurrent Neural Network
Long Short-Term Memory
Gated Recurrent Unit

Pattern Recognition & Machine Learning Laboratory
Hyeon-Woo Bae
July 21, 2021

Recurrent Neural Network (1/5)

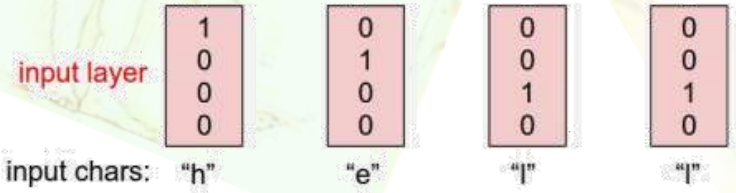
- **Recurrent Neural Network (RNN)**
  - 순차적인 정보를 인식하는데 주로 사용
    - 음성인식, 자연어처럼 연속되는 데이터 처리에 사용
  - 과거의 연산 결과가 현재 연산 결과에 영향을 미침
    - 모든 입력과 출력이 서로 독립적인 **NN, CNN**과는 다름
  - 순환되는 구조를 이용하여 과거의 학습을 가중치(**weight**)를 통해 현재 학습에 반영
  - 계산식
    - $\boldsymbol{h_t = f_W(h_{t-1}, x_t) = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)}$
    - $\boldsymbol{y_t = W_{hy}h_t}$
    - 표기
      - $\boldsymbol{x_t}$: 입력 데이터
      - $\boldsymbol{h_t}$: **t step**에서의 상태
      - $\boldsymbol{f_W}$: **RNN** 함수
      - $\boldsymbol{W}$: 가중치
      - $\boldsymbol{y_t}$: 출력 데이터
  - **RNN** 함수와 가중치는 모든 **step**에서 동일함



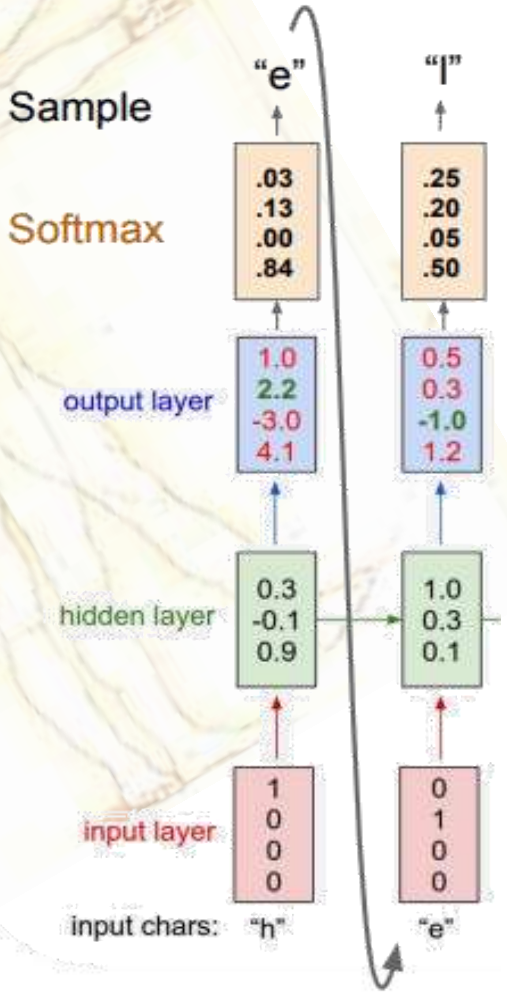
RNN의 구조

Recurrent Neural Network (2/5)

- **Recurrent Neural Network (RNN) 동작 예시**
  - ➢ 문자 기반 언어 모델
    - **One-hot encoding**을 통해 입력된 언어를 벡터로 변환
      - **h** → (1 0 0 0)
      - **e** → (0 1 0 0)
      - **l** → (0 0 1 0)
      - **l** → (0 0 1 0)
    - 변환된 벡터를 은닉층에 입력
      - $h_1 = f_W(h_0, x_1) = \tanh(W_{xh}x_1)$
      - $h_t = f_W(h_{t-1}, x_t) = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$
    - 은닉층에서 출력층으로의 계산
      - $y_t = W_{hy}h_t$
    - 출력층에서 **Softmax**함수를 이용하여 **Target** 언어로 변환
    - 출력된 언어를 다시 입력하며 학습



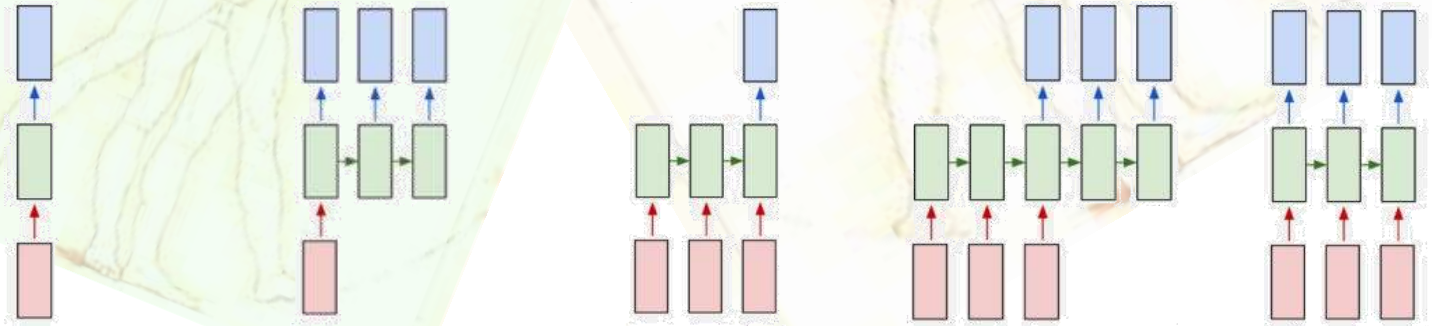
입력된 언어 변환 과정



문자 기반 언어 모델

Recurrent Neural Network (3/5)

- **Recurrent Neural Network (RNN)의 활용**
  - RNN 모델의 종류
    - 일대일 기반 모델
      - 시계열 데이터 학습
      - 음성인식
    - 일대다 기반 모델
      - 이미지 설명 생성 : 이미지를 입력하여 문자열로 출력
    - 다대일 기반 모델
      - 감정 분류 : 문자열을 입력하여 감정을 출력
    - 다대다 기반 모델
      - 기계 번역 : 문자열을 입력하여 문자열을 출력
      - 영상 분류 : 여러 프레임을 입력하여 각각 문자열을 출력



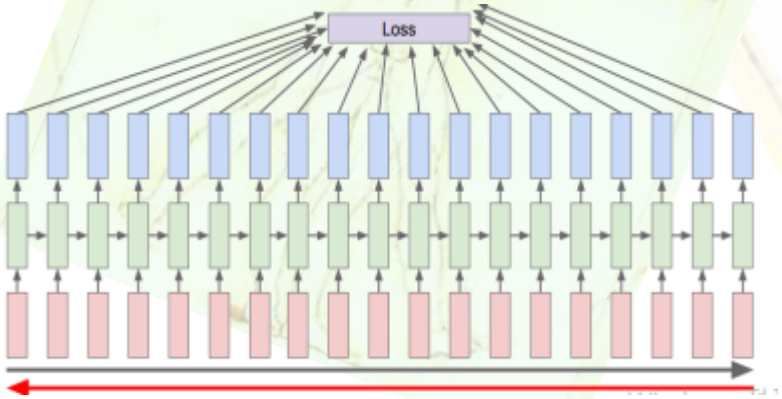
일대일 기반 모델 일대다 기반 모델 다대일 기반 모델 다대다 기반 모델

Recurrent Neural Network (4/5)

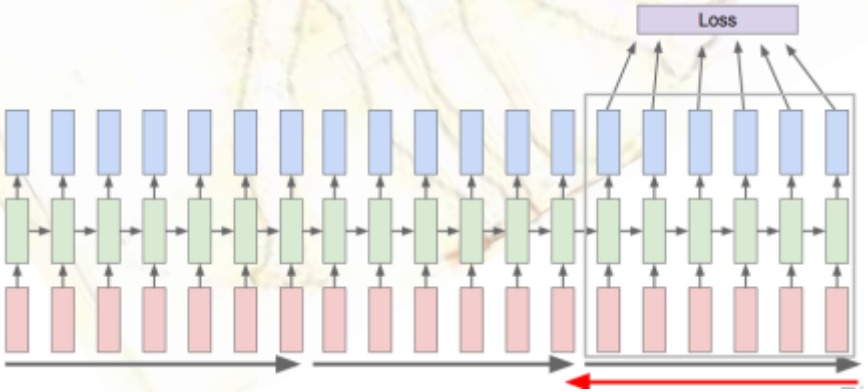
- **Recurrent Neural Network (RNN) 의 한계**
  - **장기 의존성 문제(Long-Term Dependency)**
    - **RNN**이 이전 정보를 현재 상태에서 활용
    - 비교적 최근 정보를 활용하는 작업에는 유용
    - 먼 과거 시점의 정보를 활용해야 하는 문제에서는 취약함
      - **Gradient vanishing** 때문
  - **Gradient vanishing 문제**
    - **Chain Rule**에 의해 과거의 **gradient**는 곱셈으로 표현됨
    - **Gradient**의 값이 **sigmoid** 함수나 **tanh** 의 경우 **0**에 가까움
      - **Sigmoid** 대신 **ReLU** 사용, 가중치 초기값 설정, 정규화
      - **Long Short-Term Memory(LSTM), Gated Recurrent Unit(GRU)** 구조 활용
  - **Gradient exploding 문제**
    - **Gradient**의 값이 클 경우 연속된 곱셈으로 발산하게 됨
    - **Scale**을 조정하는 **Gradient clipping**으로 해결

Recurrent Neural Network (5/5)

- **Recurrent Neural Network (RNN) 의 학습**
  - **BackPropagation Through Time (BPTT)**
    - **RNN**은 역전파 과정에서 출력층에서의 출력값이 그 전 스텝의 가중치에 영향을 미침
    - 기존의 신경망과는 달리 시간 스텝별로 네트워크를 펼친 다음 역전파 알고리즘을 사용
    - 은닉층과 출력층 사이의 가중치는 기존의 역전파와 같은 형태로 갱신
    - 입력층과 은닉층, 은닉층과 은닉층 사이의 가중치는 일정 시간 동안 에러 값들을 합친 값을 역전파하여 갱신한다.
    - **BPTT** 또한 **RNN**이 깊어질수록 **Gradient vanishing, exploding** 문제에 직면함
  - **Truncated BackPropagation Through Time**
    - 계산량을 줄이기 위해 현재 스텝에서 일정 시간 이전까지만 에러를 역전파하여 갱신



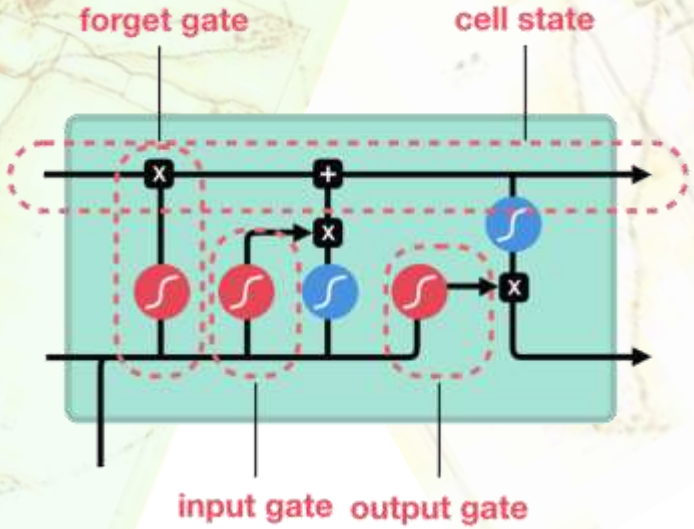
BPTT의 가중치 갱신 알고리즘



Truncated BPTT의 가중치 갱신 알고리즘

Long Short-Term Memory

- **Long Short-Term Memory(LSTM)**의 특징
  - **RNN**의 **Gradient vanishing** 문제를 해결하고자 고안됨
  - **RNN**의 은닉층에 **Cell State**를 추가한 구조
  - 과거의 정보를 꾸준히 업데이트하여 장기 의존성 문제를 해결
  - 가중치를 덧셈으로 처리하기 때문에 **Gradient vanishing** 문제를 해결
  - 메모리가 덮어씌워질 가능성이 있고 연산 속도가 느림

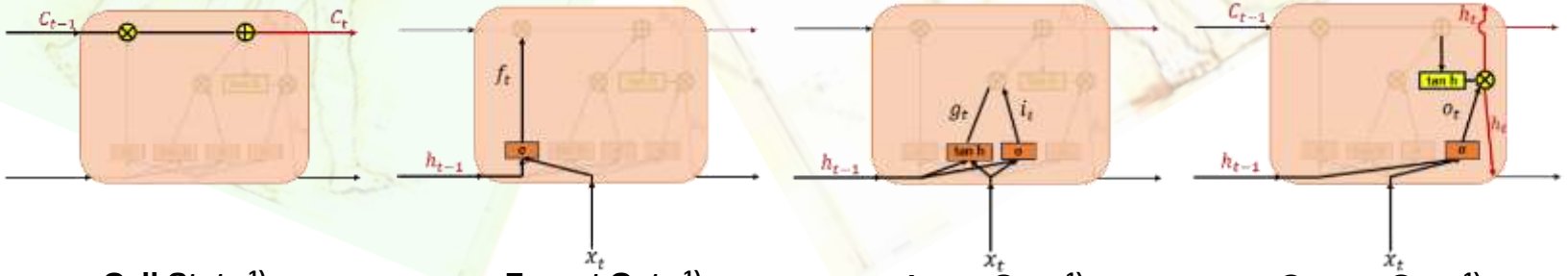


LSTM의 구조 [1)]

1) https://www.programmersought.com/article/969975979/

Long Short-Term Memory

- **Long Short-Term Memory(LSTM)의 구조 및 동작 방법**
  - **Cell State**
    - Hidden State와 마찬가지로 이전 시점의 정보를 다음 Cell State로 전달
    - Gate들과 함께 작용하여 정보를 선택적으로 활용
  - **Forget Gate**
    - 기존 정보를 기억할 것인지 버릴 것인지 결정
    - 0에 가까울수록 많은 정보를 잊고 1에 가까울수록 많은 정보를 기억
  - **Input Gate**
    - 현재 정보를 기억할 것인지 버릴 것인지 결정
    - 현재 정보를 어떤 비율로 더할지 결정
  - **Output Gate**
    - 어떤 정보를 출력할지 결정

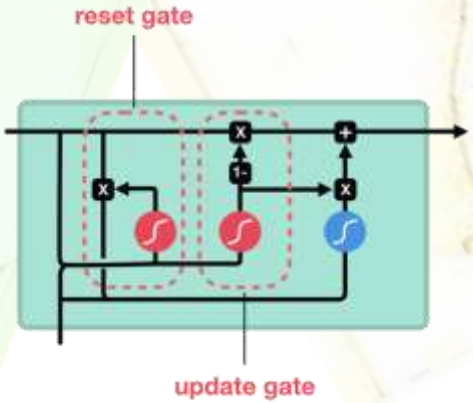


Cell State [1)] Forget Gate [1)] Input Gate [1)] Output Gate [1)]

1) https://wikidocs.net/60762

Gated Recurrent Unit

- **Gated Recurrent Unit(GRU)의 구조**
  - 기존 **LSTM**의 구조를 간단하게 개선한 모델
    - **Cell State**와 **Hidden State**가 합쳐져 하나의 **Hidden State**로 구성
    - **Reset Gate**
      - 과거시점의 **hidden state**와 현 시점의 입력값을 시그모이드를 적용하여 연산
      - 과거시점의 정보를 얼마나 활용할 것인가에 대한 정보
    - **Update Gate**
      - 과거와 현재의 정보를 각각 얼마나 반영할 것인지에 대한 비율을 결정
- **Gated Recurrent Unit(GRU)의 특징**
  - 메모리가 덮어씌워질 가능성이 없음
  - 성능면에서 LSTM과 유사하나 학습할 파라미터가 더 적어 연산 속도가 빠름



GRU의 구조 [1)]

1) https://www.programmersought.com/article/969975979/