Machine Learning Study

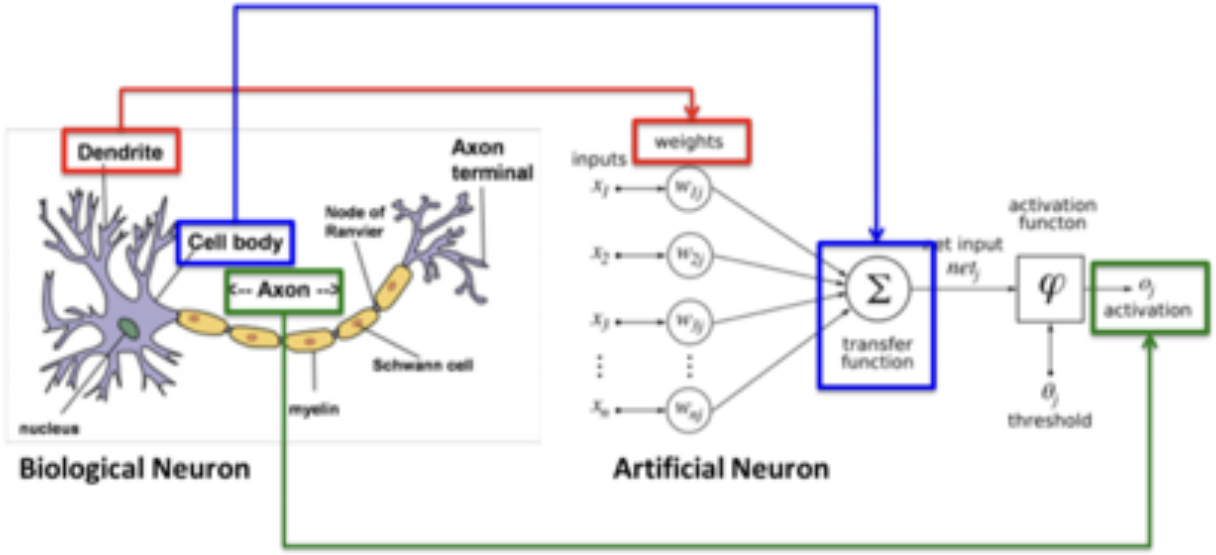
-Part 2

지난 시간 복습

- 신경망과 퍼셉트론(perceptron)

퍼셉트론 (1958)

- Rosenblatt은 신경망 모델의 일종인 퍼셉트론과 그 학습 알고리즘을 제시



- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

지난 시간 복습

- 신경망과 퍼셉트론(perceptron)

  - **퍼셉트론 구조**
    - 생물학적 신경세포의 작동 방식을 **수학적으로 모델링**한 **인공 뉴런**
    - 뇌의 구조를 모사한 정보처리 방식
      - 1958년 Rosenblatt가 학습이 가능함을 증명
    - 인접한 두 뉴런의 연결부분인 **시냅스**에서 학습이 일어남
    - 출력노드의 활성화값

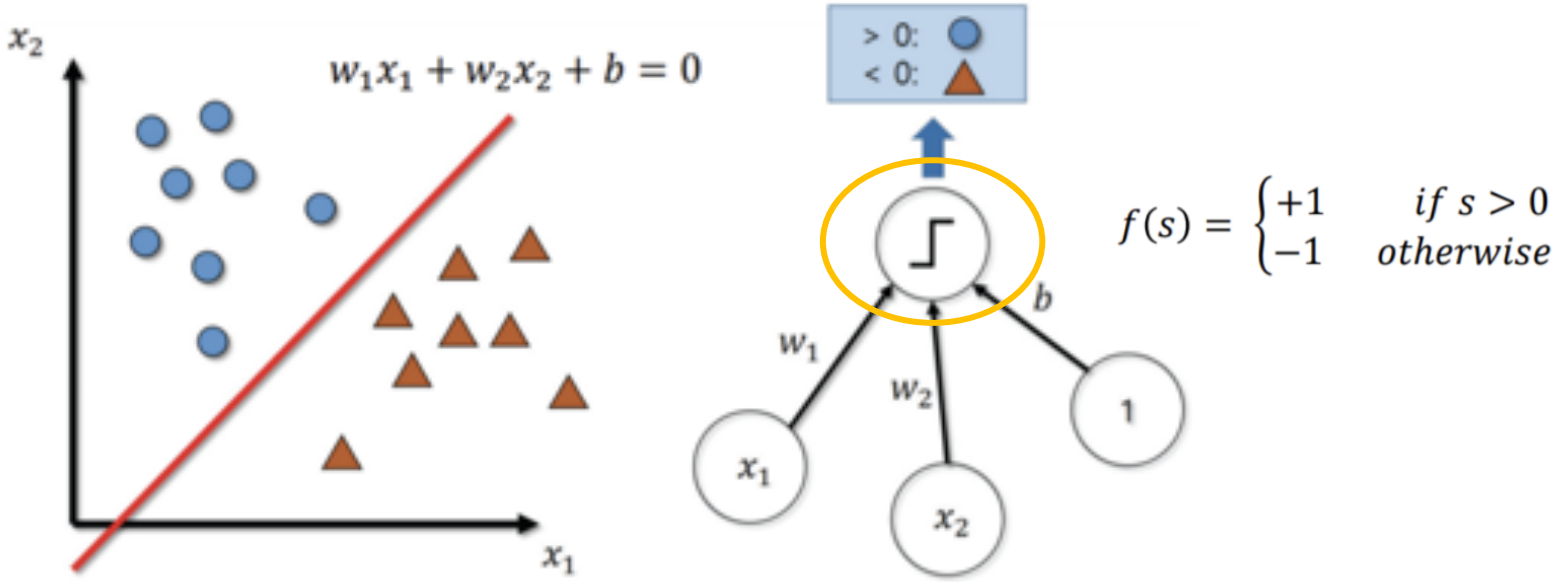
$$s = \sum_{i=1}^{n} w_i x_i + w_0 = \sum_{i=0}^{n} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

    - 임계논리 활성화 함수: 임계치를 넘으면 +1을, 넘지 않으면 -1을 출력
      - Rosenblatt의 퍼셉트론(단순 퍼셉트론)은 하나의 임계논리유닛으로 구성

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

지난 시간 복습

- 신경망과 퍼셉트론(perceptron)
  - 단순 퍼셉트론의 작동 원리



- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

지난 시간 복습

- 신경망과 퍼셉트론(perceptron)
  - 퍼셉트론의 학습
    - 학습 목표: 주어진 학습패턴들을 두 개의 클래스로 분류하는 **경계면 찾기**

$$f(s) = \begin{cases} +1 & if\ s > 0 \\ -1 & otherwise \end{cases}$$

    - 학습 패턴 x가 입력될 때 출력되는 f를 계산, 목표 출력과 일치하지 않으면 변경
    - 퍼셉트론 학습 규칙
      - **학습예를 반복적으로 관측하면서 가중치를 변경하면 선형 분리가 가능한 분류 문제의 경우 올바른 분류기를 학습할 수 있음**

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

지난 시간 복습

- 신경망과 퍼셉트론(perceptron)

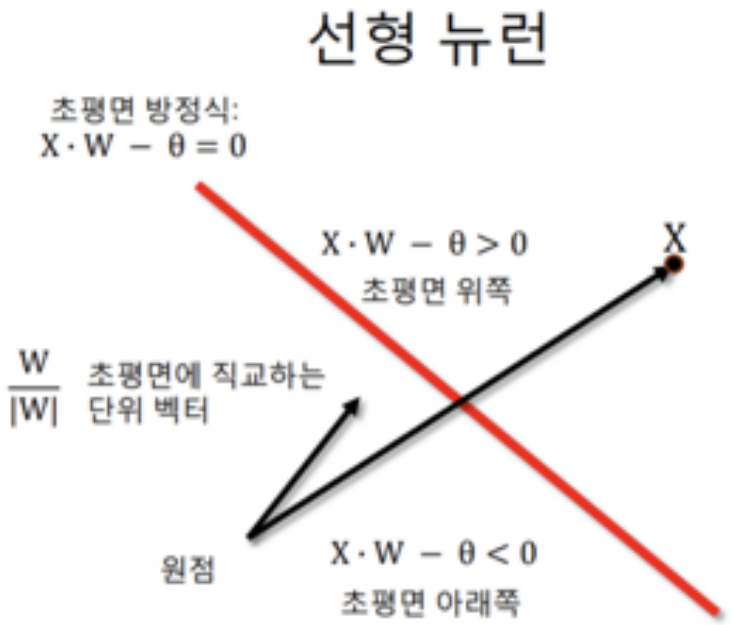
  - 단순 퍼셉트론의 학습 원리

시작:
가중치 벡터 w 를 랜덤하게 생성

테스트:
벡터 $x \in P \cup N$ 를 랜덤하게 선택
$x \in P$ 이고 $w \cdot x > 0$ 이면 goto 테스트,
$x \in P$ 이고 $w \cdot x \geq 0$ 이면 goto 더하기,
$x \in N$ 이고 $w \cdot x < 0$ 이면 goto 테스트,
$x \in N$ 이고 $w \cdot x \leq 0$ 이면 goto 빼기.

더하기:
$w = w + x$로 설정, goto 테스트

빼기:
$w = w - x$로 설정, goto 테스트



- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

지난 시간 복습

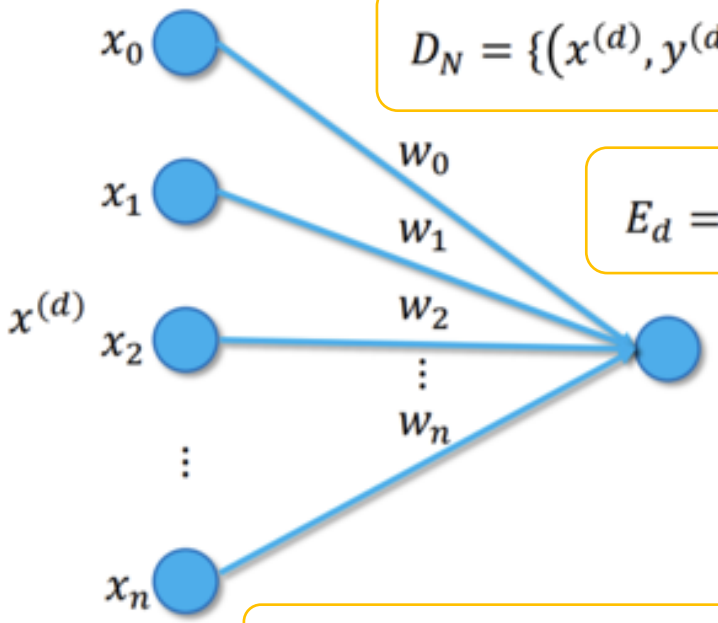
- 단순 퍼셉트론(single perceptron)의 학습, 델타규칙
  - 임계활성화함수의 한계
    - +1 또는 -1의 이진값만을 출력
    - 오차는 0 아니면 2만 가능: 미세한 오차 조정 불가능
  - 선형 활성화 함수
    - 임계활성화함수의 한계를 해결
    - 선형퍼셉트론: 선형 활성화 함수를 사용하는 퍼셉트론
    - 델타 규칙을 사용하여 학습 가능 (Widrow & Hoff, 1960)

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

지난 시간 복습

- 단순 퍼셉트론(single perceptron)의 학습, 델타규칙

■ 선형퍼셉트론의 구조



$D_N = \{(x^{(d)}, y^{(d)})\}_{d=1}^{N}$

학습 데이터 N개

$E_d = \frac{1}{2}(y^{(d)} - f^{(d)})^2$

오차 계산(E_d)

$f^{(d)} \Leftrightarrow y^{(d)}$

$f^{(d)}$: 출력 값
$y^{(d)}$: 학습데이터 y

$f^{(d)} = f(x^{(d)}; w) = \sum_i w_i x_i$

w_i: 가중치
가중치들을 학습!!

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

지난 시간 복습

- 단순 퍼셉트론(single perceptron)의 학습, 델타규칙
  - 델타규칙에 의한 학습 알고리듬(cont.)
    - **오차를 줄여주는 방향**으로 시냅스 가중치를 교정

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = -\eta \frac{\partial E_d}{\partial w_i}$$

오차를 가중치 w_i에 대해서 미분

$$\frac{\partial E_d}{\partial w_i} = \frac{\partial E_d}{\partial f^{(d)}} \frac{\partial f^{(d)}}{\partial w_i} = \frac{\partial}{\partial f^{(d)}} \frac{1}{2} \left(y^{(d)} - f^{(d)}\right)^2 \frac{\partial f^{(d)}}{\partial w_i}$$

$$= \frac{1}{2}(-2)\left(y^{(d)} - f^{(d)}\right) x_i^{(d)} = -\left(y^{(d)} - f^{(d)}\right) x_i^{(d)}$$

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

지난 시간 복습

- 단순 퍼셉트론(single perceptron)의 학습, 델타규칙

  - 델타규칙에 의한 학습 알고리듬(cont.)
    - i번째 가중치의 학습식
      - $\eta \in (0,1)$ 은 학습률 (learning rate)

$$w_i \leftarrow w_i + \eta\left(y^{(d)} - f^{(d)}\right)x_i^{(d)}$$

- 델타규칙을 바탕으로 모멘텀을 추가하거나 속도, 가속도 등을 사용하여 학습을 시키는 방법들
- Adam optimizer, SGD optimizer 등

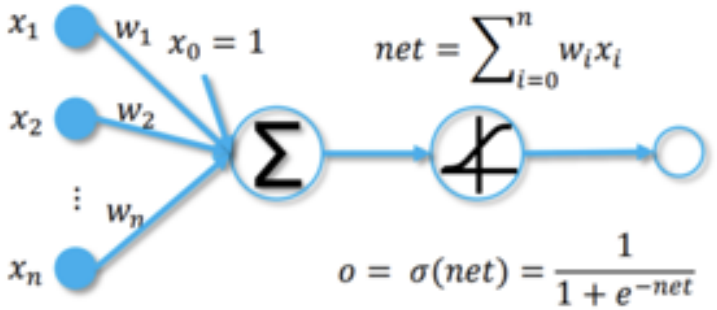
- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

지난 시간 복습

- 단순 퍼셉트론(single perceptron)의 학습, 시그모이드 함수의 등장(1986년)

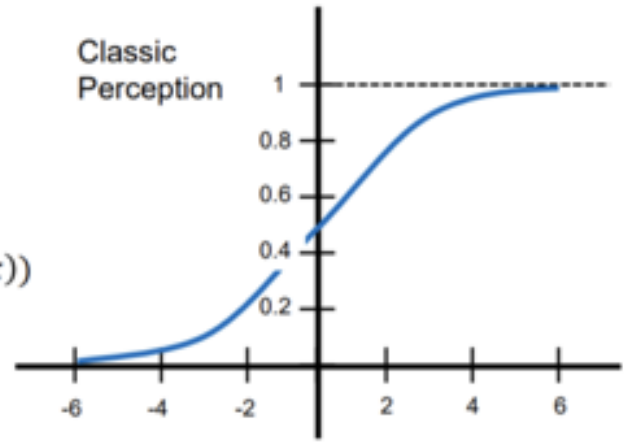
■ 시그모이드 함수 뉴런

- 선형퍼셉트론은 **선형 분리**가 가능한 패턴분류 문제만 해결 가능
- 복잡한 문제를 풀기 위해서는 **비선형 결정 경계**를 생성해야 함
- 미분이 가능하려면 연속함수일 필요가 있음
- **시그모이드(sigmoid)** 함수: 비선형, 미분 가능한 연속함수



Sigmoid function is **Differentiable**

$$\frac{df(x)}{dx} = f(x)(1-f(x))$$



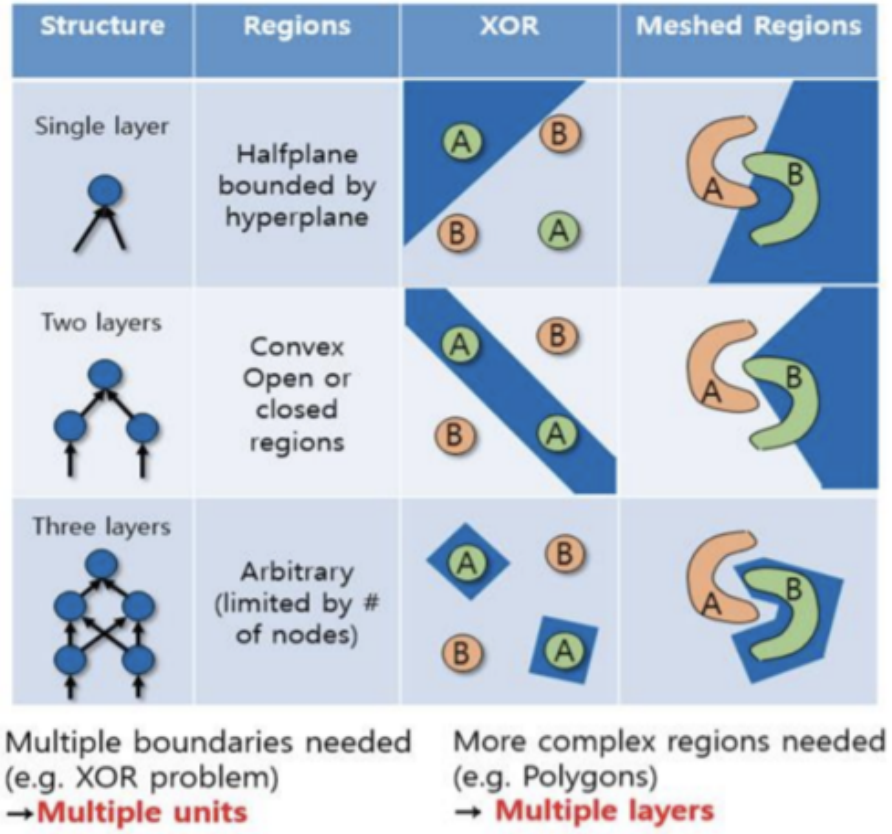
- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

지난 시간 복습

- 다층 퍼셉트론(Multi-Layer Perceptron a.k.a. MLP)
- 다수, 다층 뉴런의 필요성

- 다수 뉴런
- 뉴런이 하나: 하나의 구분선
- 뉴런이 둘: 두 개의 구분선

- 다층 뉴런
- Dimension이 증가하게 되면 구분에 Multiple layer가 필요

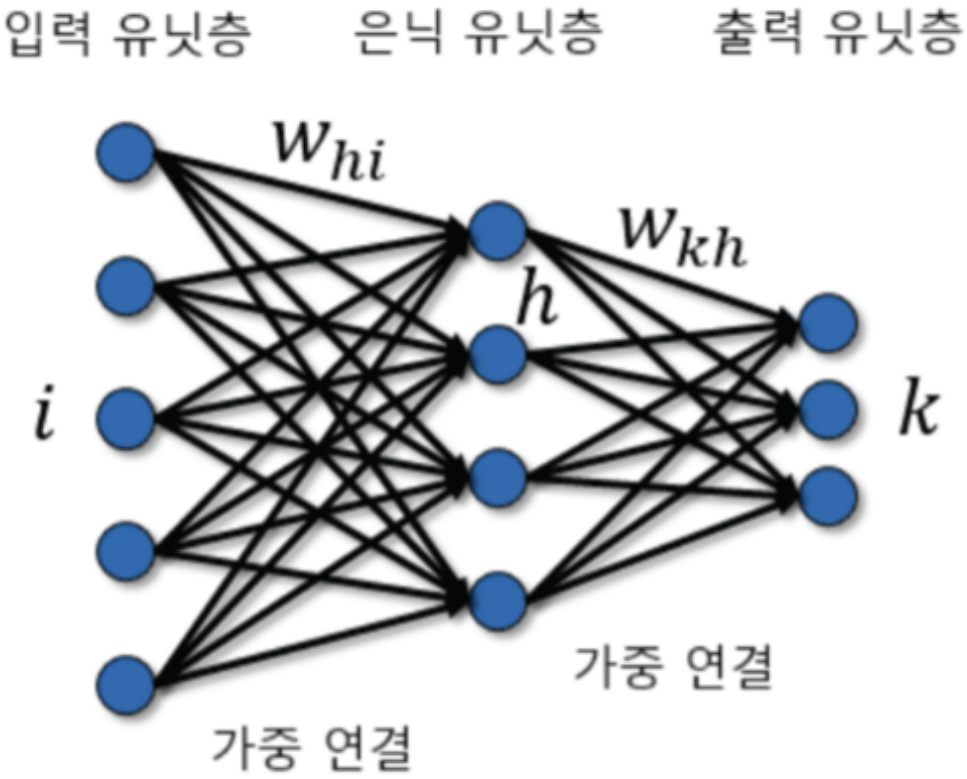


- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

지난 시간 복습

- 다층 퍼셉트론(Multi-Layer Perceptron. MLP)

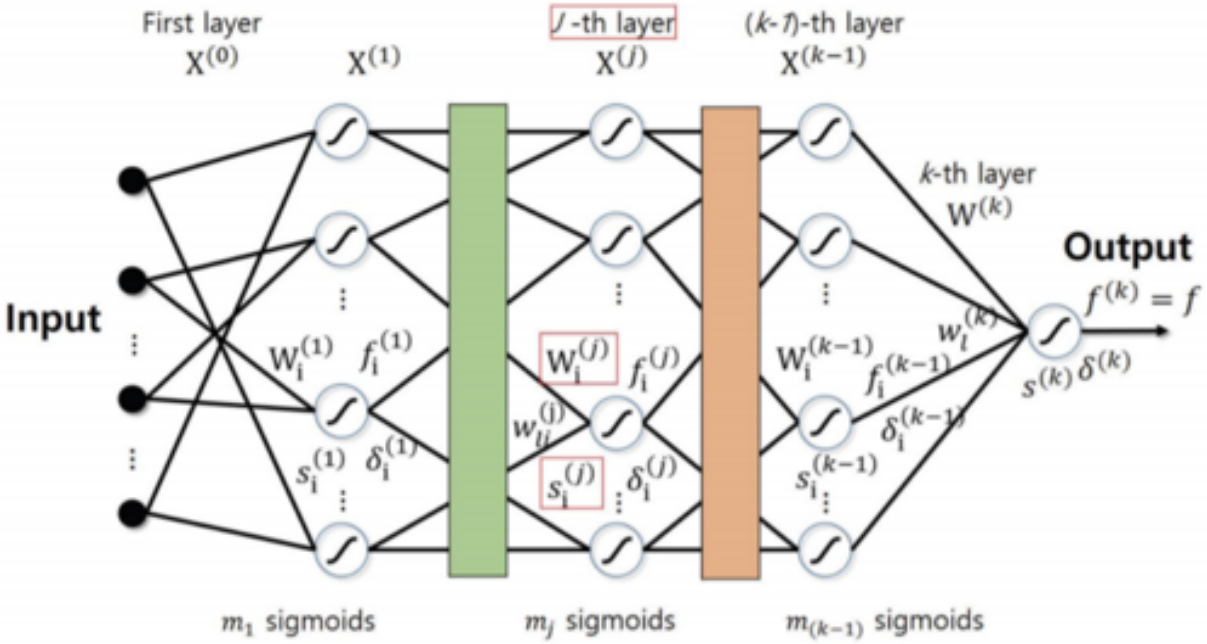
- 입력층, 은닉층, 출력층
- 각각의 뉴런은 이전의 뉴런과 모두 연결
- Fully Connected Network(FCN)
- 입력과 출력이 데이터셋과 동일하도록 Layer 사이의 연결, weight를 학습하는 것이 목표
- MLP를 I-H-K layer로 표현하기도 함
  (I, H, K – I개 input neuron, h개 hidden neuron, k개 출력 뉴런)



- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

지난 시간 복습

- 다층 퍼셉트론(Multi-Layer Perceptron. MLP)
  - 다층퍼셉트론의 혁신
    - 시그모이드 뉴런의 사용
      - 비선형 모델로 확장 가능
      - 연속함수이므로 미분 가능
      - 은닉 뉴런층의 학습을 가능하게 하는 오류역전파 알고리듬의 개발로 이어짐
    - **은닉 뉴런층** 추가
    - 은닉층 학습방법 도입
- 1986년 이전(sigmoid 함수 등장 이전)에는 다층의 weight을 학습할 방법이 부재
- Sigmoid 함수의 등장,
- Back propagation



- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

지난 시간 복습

- 오류 역전파(Back Propagation)
- 계산 그래프를 이용한 개념 정리: 순전파

문제 1: 현빈 군은 슈퍼에서 1개에 100원인 사과를 2개 샀습니다. 이때 지불 금액을 구하세요, 단 소비세가 10% 부과됩니다.

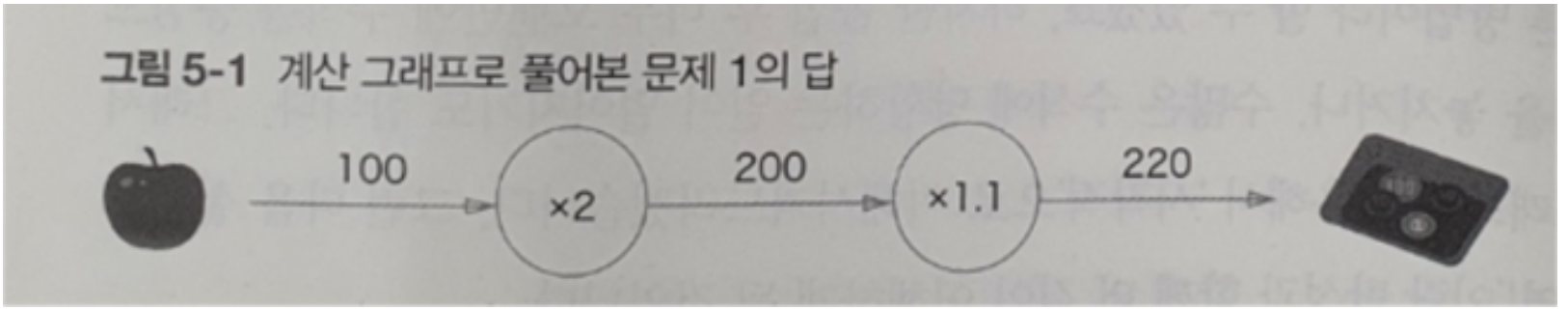


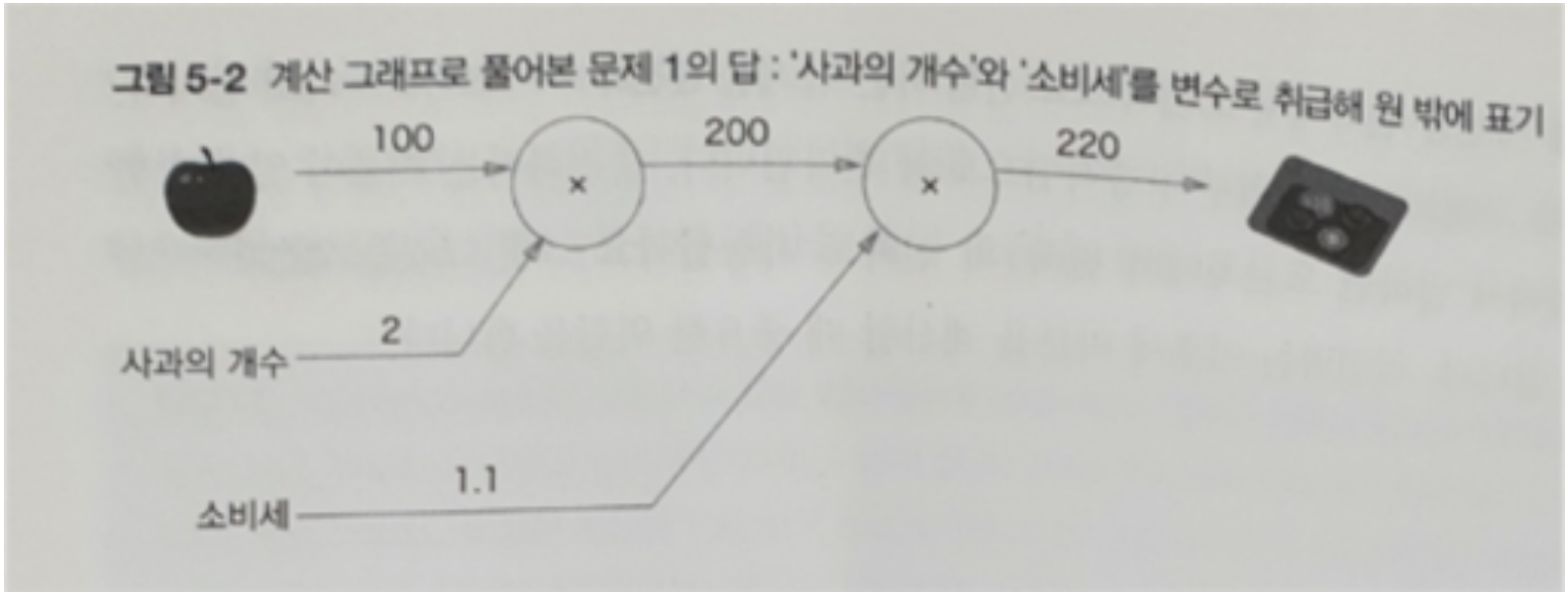
그림 5-1 계산 그래프로 풀어본 문제 1의 답

- 밑바닥부터 시작하는 딥러닝 (2017) 사이토 고키 저 참조

지난 시간 복습

- 오류 역전파(Back Propagation)
- 계산 그래프를 이용한 개념 정리: 순전파

문제 1: 현빈 군은 슈퍼에서 1개에 100원인 사과를 2개 샀습니다. 이때 지불 금액을 구하세요, 단 소비세가 10% 부과됩니다.

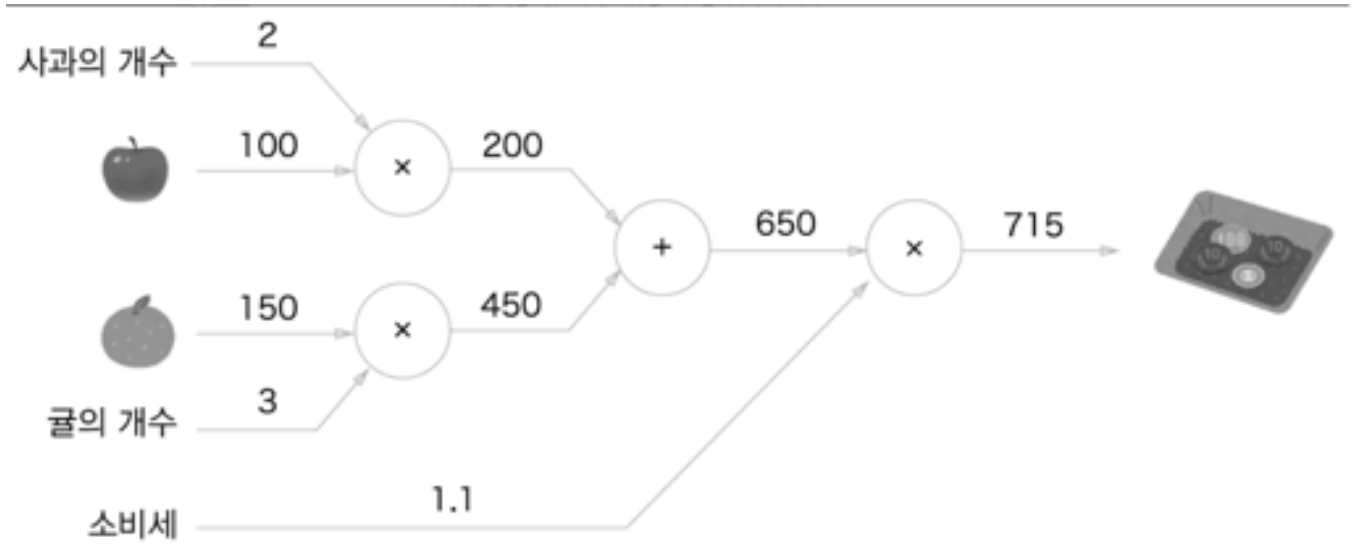


- 밑바닥부터 시작하는 딥러닝 (2017) 사이토 고키 저 참조

지난 시간 복습

- 오류 역전파(Back Propagation)

  문제 2: 현빈 군은 슈퍼에서 사과를 2개, 귤을 3개 샀습니다. 사과는 1개에 100원, 귤은 1개 150원입니다. 소비세가 10%일 때, 지불 금액을 구하세요.

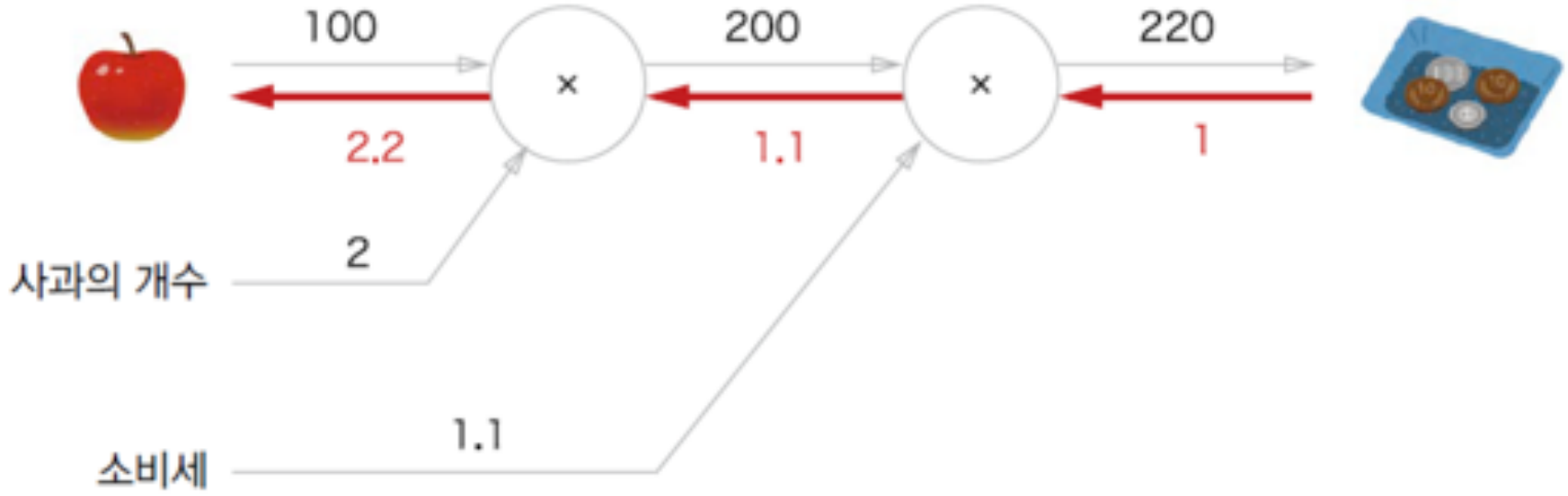


- 밑바닥부터 시작하는 딥러닝 (2017) 사이토 고키 저 참조

지난 시간 복습

- 오류 역전파(Back Propagation)

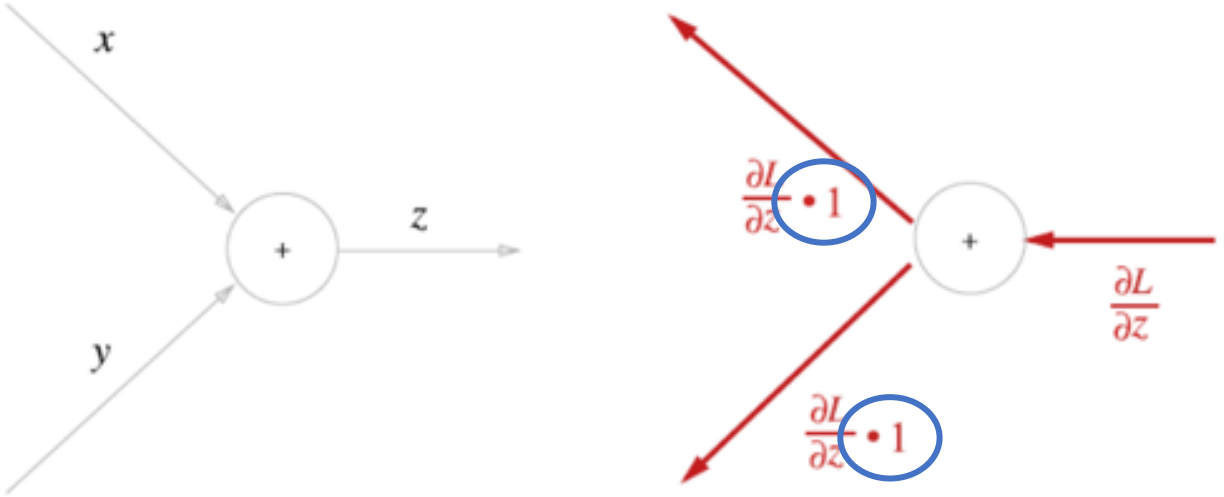
  문제 1: 현빈 군은 슈퍼에서 1개에 100원인 사과를 2개 샀습니다. 이때 지불 금액을 구하세요, 단 소비세가 10% 부과됩니다.



- 밑바닥부터 시작하는 딥러닝 (2017) 사이토 고키 저 참조

지난 시간 복습

- 오류 역전파(Back Propagation)
- 덧셈 노드의 역전파

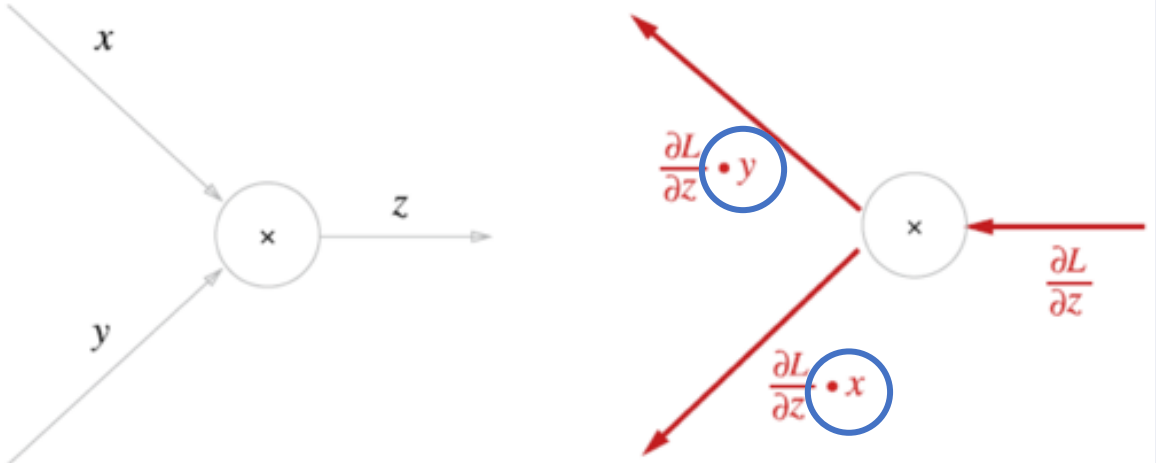


- x + y = z
- z값에 대하여 편미분한 값이 전파
- $\frac{\partial z}{\partial y} = 1, \frac{\partial z}{\partial x} = 1$
- 각각 1을 곱한 값을 역전파

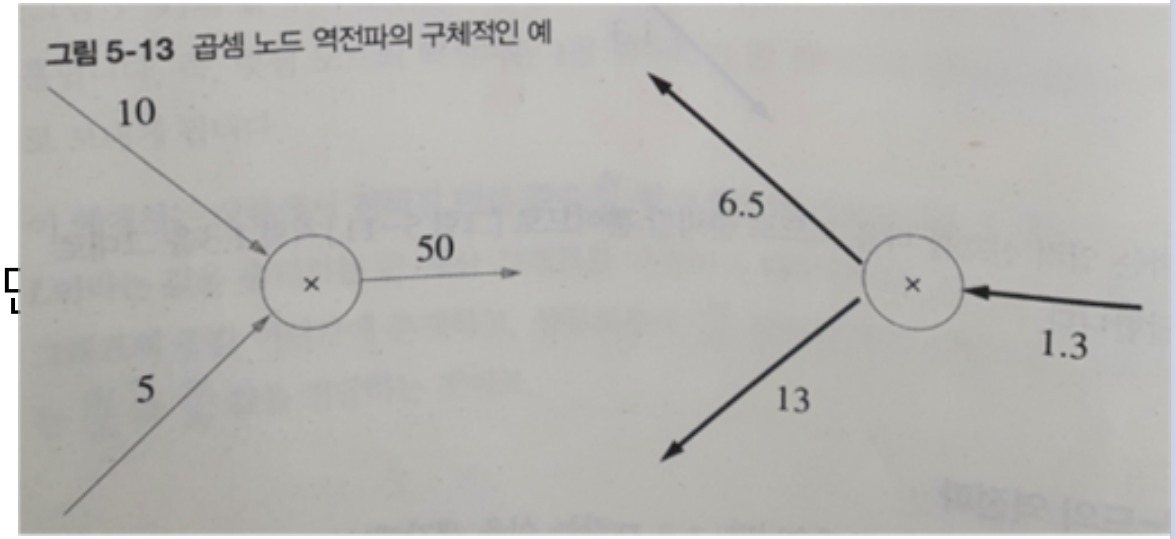
- 밑바닥부터 시작하는 딥러닝 (2017) 사이토 고키 저 참조

지난 시간 복습

- 오류 역전파(Back Propagation)
- 곱셈 노드의 역전파
- x×y = z
- $\frac{\partial z}{\partial x}$=y, $\frac{\partial z}{\partial y}$ = x
- 각각에 y, x를 곱한 값을 역전파



- 예시: 10 ×5 = 50 에 대한 역전파
- 1.3 이라는 값이 곱셈 노드로 들어온다면
- 각각 6.5, 13에 해당하는 값을 전파

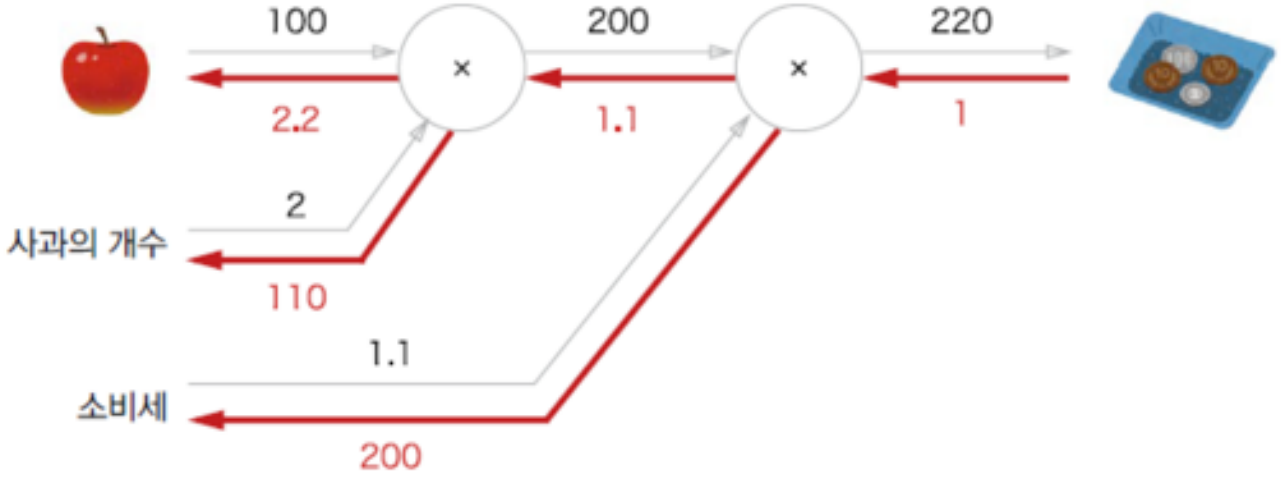


- 밑바닥부터 시작하는 딥러닝 (2017) 사이토 고키 저 참조

지난 시간 복습

- 오류 역전파(Back Propagation)

  문제 1: 현빈 군은 슈퍼에서 1개에 100원인 사과를 2개 샀습니다. 이때 지불 금액을 구하세요, 단 소비세가 10% 부과됩니다.

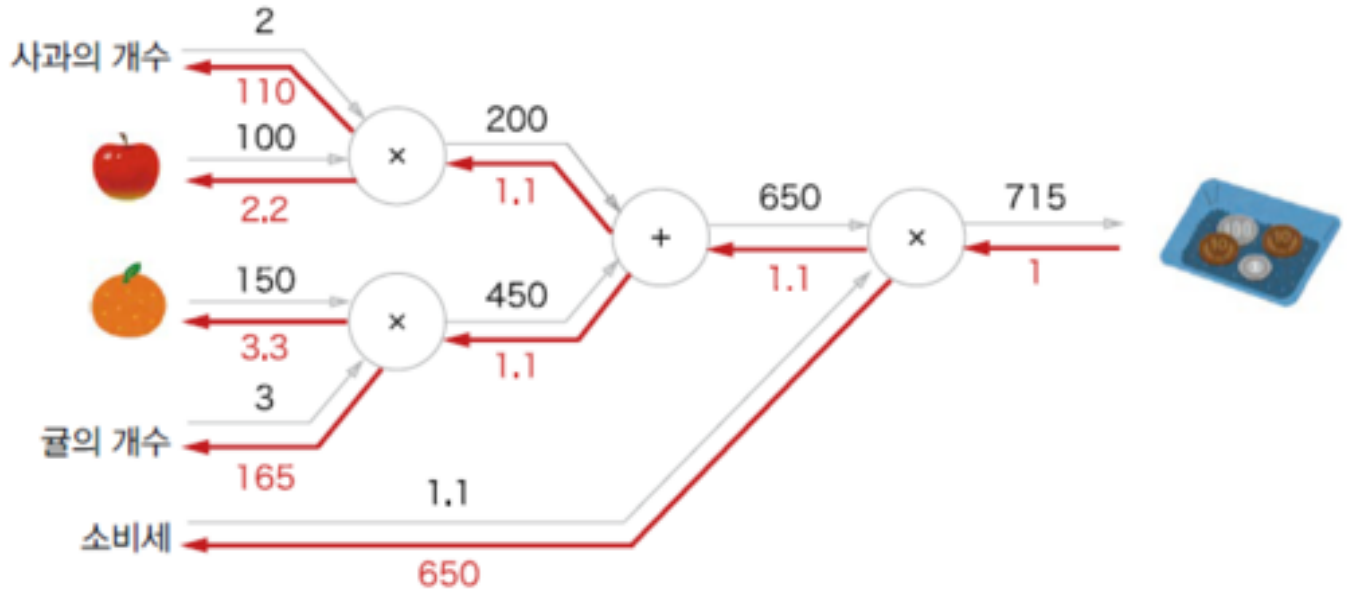


- 밑바닥부터 시작하는 딥러닝 (2017) 사이토 고키 저 참조

지난 시간 복습

- 오류 역전파(Back Propagation)

  문제 2: 현빈 군은 슈퍼에서 사과를 2개, 귤을 3개 샀습니다. 사과는 1개에 100원, 귤은 1개 150원입니다. 소비세가 10%일 때, 지불 금액을 구하세요.



- 밑바닥부터 시작하는 딥러닝 (2017) 사이토 고키 저 참조

지난 시간 복습

- 오류 역전파(Back Propagation)

  뉴런을 학습할 때에는 거의 모든 경우 행렬로 연산

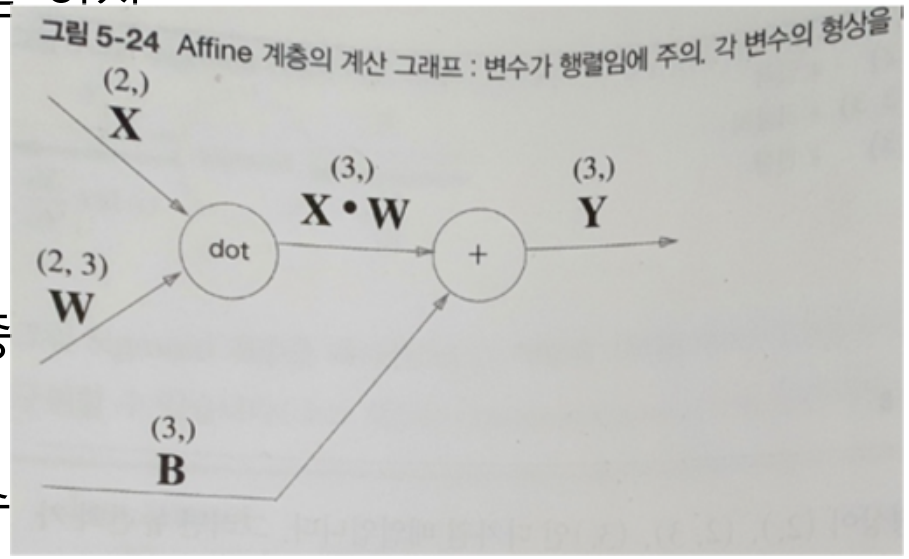
  오류 역전파는 행렬로 이루어져야함

  (Y = W·X + B)

  Affine 계층이란: 실제 계산이 이루어 지는 층

  (Y= W·X + B)

  Affine 계층을 통과한 Y는 이후 sigmoid 함수 Softmax 함수, ReLU함수 등을 통과하여 다음 Layer로 전달



- 밑바닥부터 시작하는 딥러닝 (2017) 사이토 고키 저 참조

지난 시간 복습

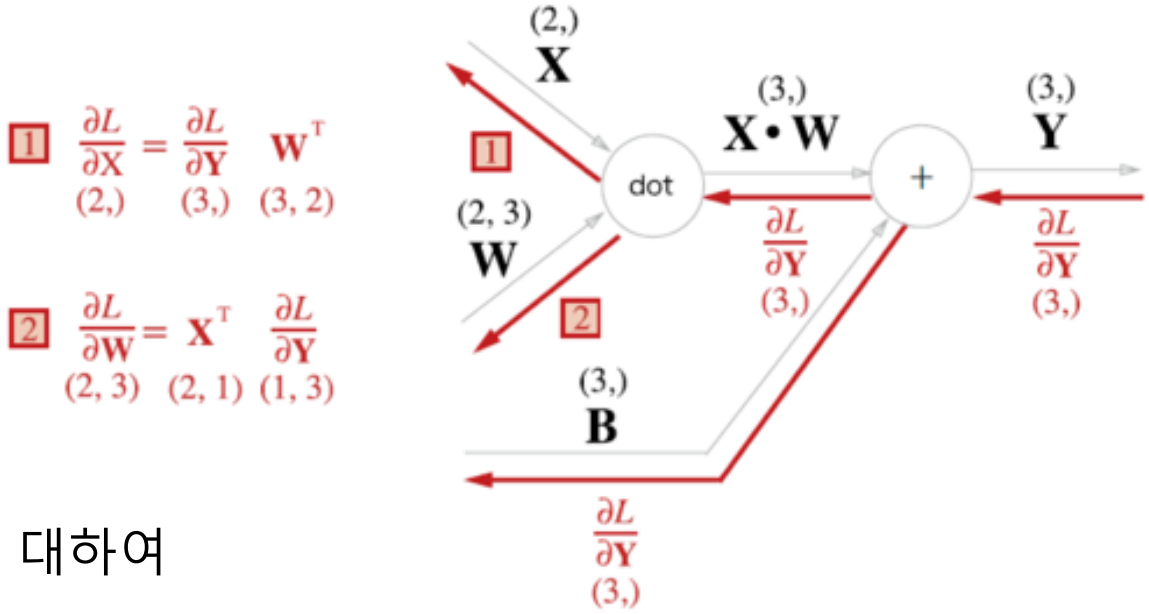
- 오류 역전파(Back Propagation)

역전파를 위해 편미분을 해보자

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \cdot W^T$$

$$\frac{\partial L}{\partial W} = X^T \cdot \frac{\partial L}{\partial Y}$$

각각에 편미분한 행렬을 역전파



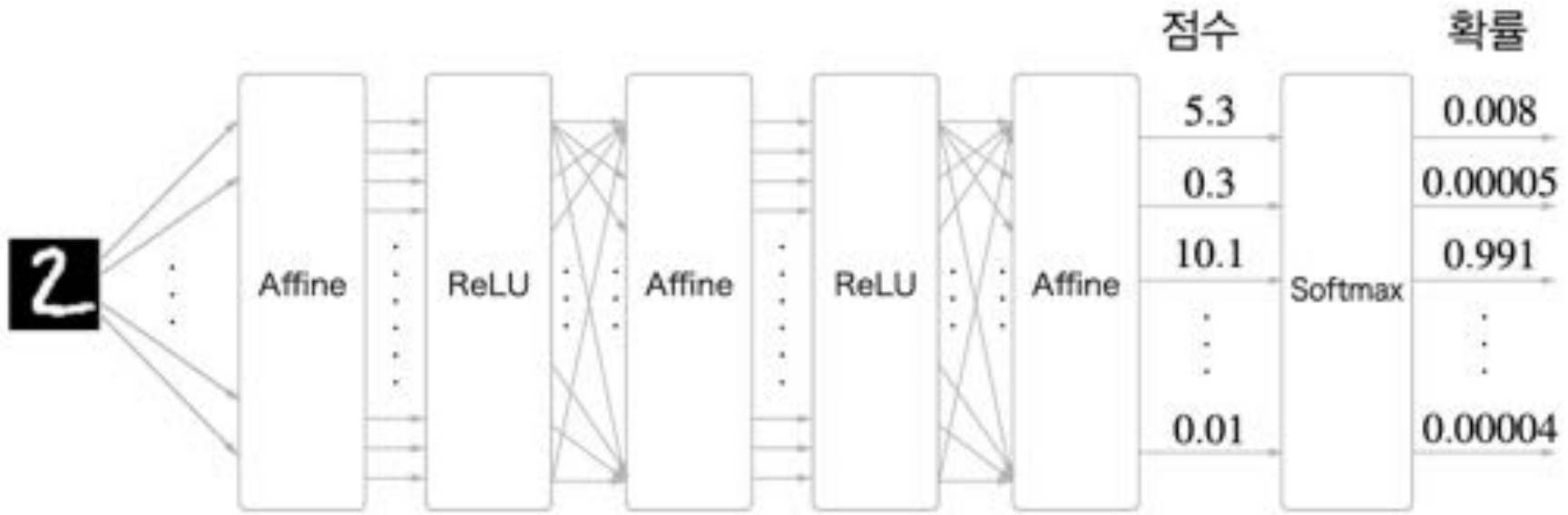
머신러닝에서 역전파란 발생한 오차에 대하여
미분한 값을 계속 뒤로 전달하는 것
전파되는 값은 커지지 않음: 중간에 softmax 함수 등 활성화 함수를 통과하며 값이 줄어들기 때문

- 밑바닥부터 시작하는 딥러닝 (2017) 사이토 고키 저 참조

지난 시간 복습

- 오류 역전파(Back Propagation)

실제 layer들의 구성: Affine 계층과 Activation 계층의 반복



- 밑바닥부터 시작하는 딥러닝 (2017) 사이토 고키 저 참조

지난 시간 복습

- 오류 역전파(Back Propagation)의 문제점과 해결
- 수렴 속도가 느림
  - Momentum term(1차 미분)
  - Hessian(2차미분) – 더 정확하지만 계산량이 증가해 계산 cost면에서 적절한 trade-off가 필요
- 과다학습(Overfitting) 문제
  - Weight Decay
  - Stochastic Descent
  - Early Stopping
- Local Minimum을 찾는 문제 – Global minimum을 찾기 어려움
  - Global searching algorithm: Simulated Annealing, Genetic Algorithm

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

지난 시간 복습

- 다층 퍼셉트론(Multi-Layer Perceptron. MLP) 활용 예시
- 다양한 감독 학습 문제에 적용 가능
  - 입력과 출력이 다차원인 경우
  - 입력과 출력이 이산값 / 연속값인 경우

- MNIST 데이터셋에 적용(LeCun, Cortes, & Burges, 1998) □
- 입력: 숫자 이미지 픽셀값의 벡터 □
- 출력: 0부터 9까지의 숫자 클래스 값

- 스팸 메일 여과기 만들기(Clark, Koprinska, & Poon, 2003) □
- 입력: 메일에 나오는 단어들에 대한 빈도수 벡터 □
- 출력: 스팸인지 아닌지를 판별하는 두 개의 클래스 값

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

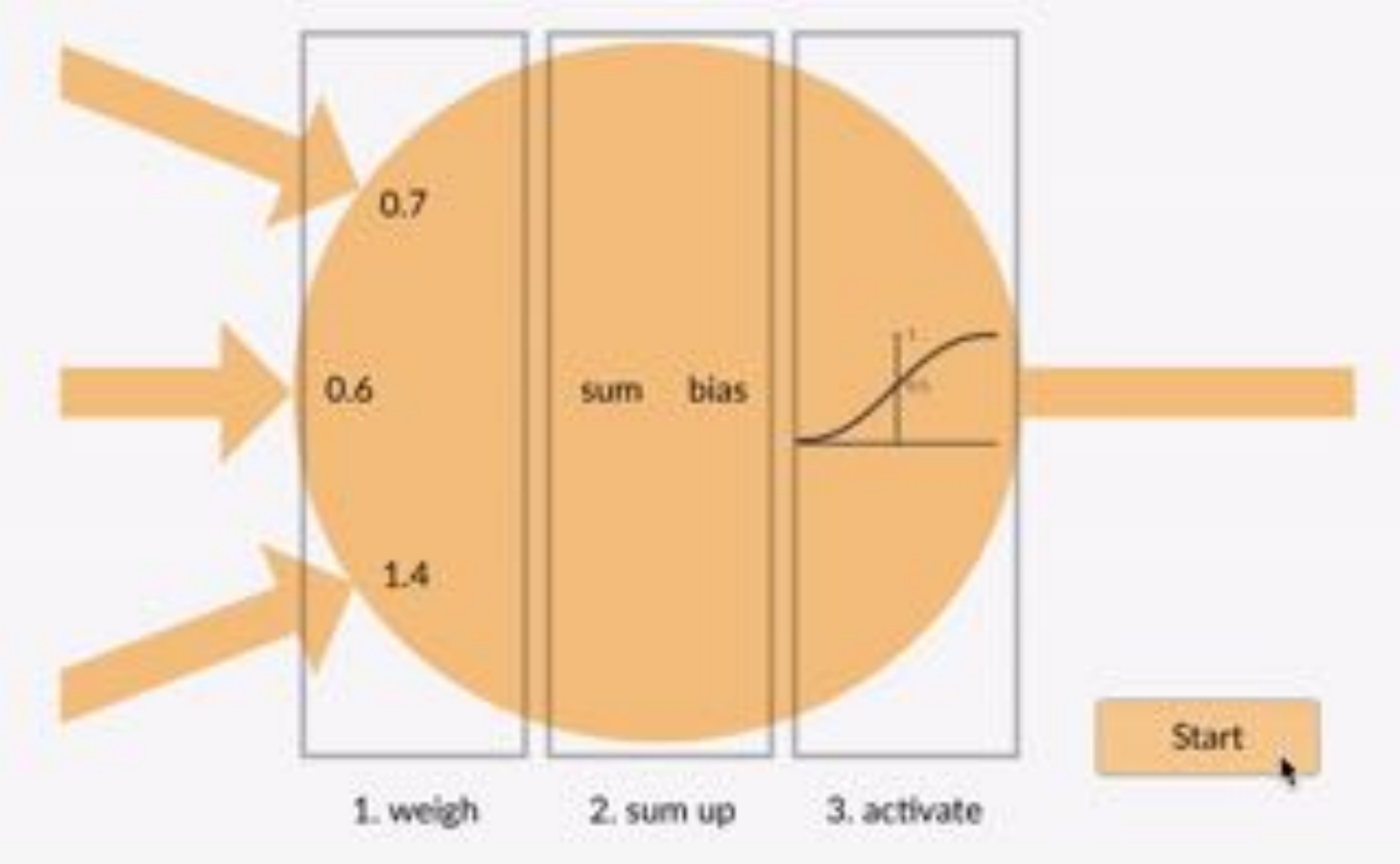
지난 시간 복습

- 요약
  - 퍼셉트론
    - 생물학적 신경세포의 작동 방식을 **수학적으로 모델링**한 **인공 뉴런**
  - 델타 규칙
    - 선형퍼셉트론의 학습을 위해 사용
  - 다층 퍼셉트론
    - 비선형 모델로 확장됨, 미분 가능: 오류역전파 알고리듬의 개발로 연결
  - 오류역전파 알고리듬
    - 상위층 뉴런의 델타값을 이용하여 하위층 뉴런의 델타값을 계산
  - 활용 사례
    - 얼굴 인식, 음성 인식 등 복잡한 패턴 인식에 응용됨

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

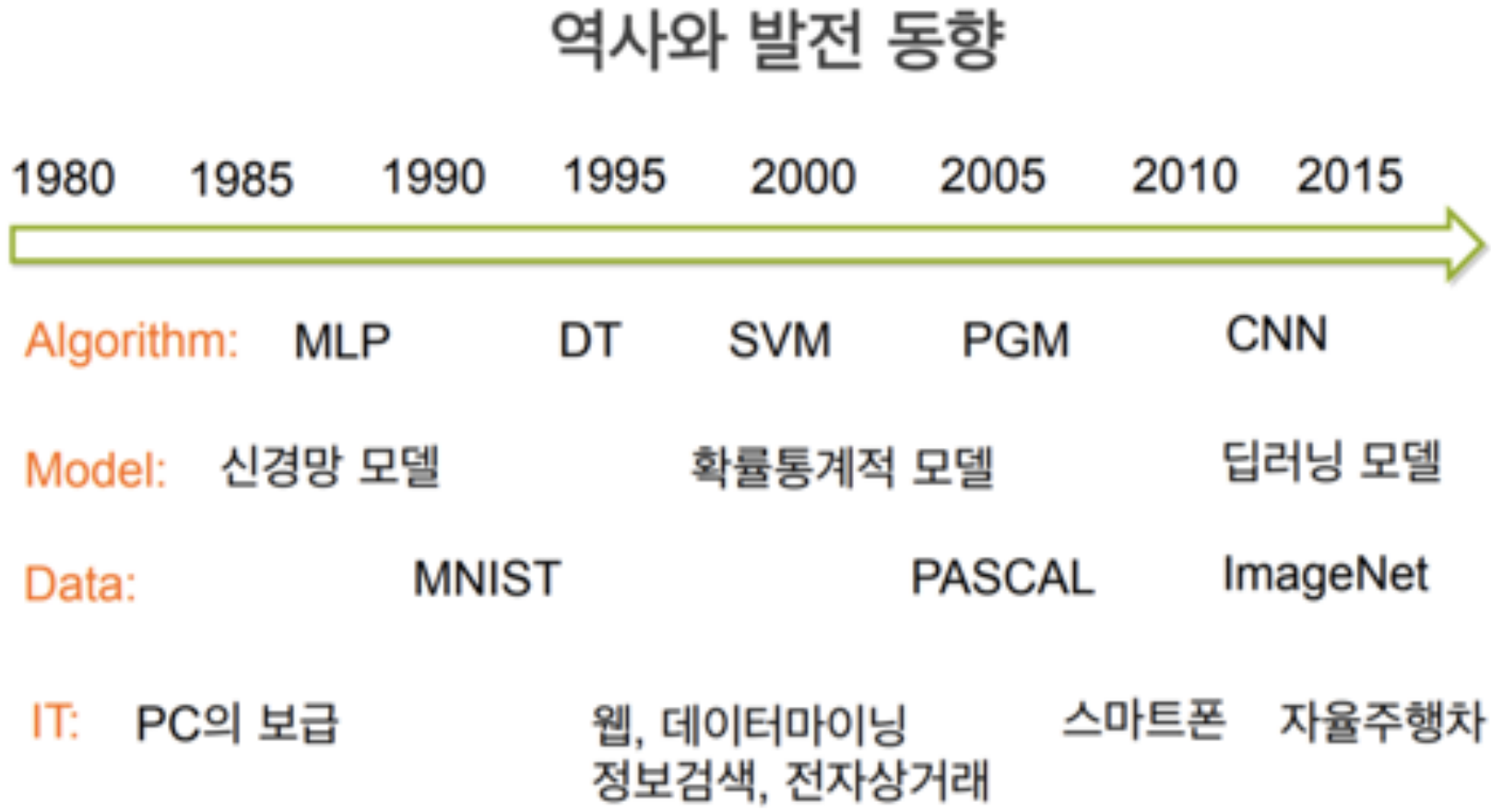
지난 시간 복습

- 요약



- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Machine Learning Study - Part2. Attention



- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Before we start, … Deep Learning and Regularization

- Deep Learning이 주목받게 된 이유?
- Deep Learning은 복잡한 문제를 풀기 위하여 사용/적당히 풀 수 있는 문제들은 SVM(support vector machine,) MLP(multi-layer perceptron)으로 충분히 풀 수 있음
- SVM, MLP로 풀 수 있는 문제들을 Deep Learning으로 풀게 된다면 Overfitting 문제가 발생

- 컴퓨팅 파워의 향상
- 학습 데이터가 많아져, 아주 많은 데이터를 학습시킴으로써 복잡한 모델구조가 과다학습되지 않음
- Deep Learning이 문제를 푸는데 매우 유용(ex. AlexNet (2012))

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Before we start, ... Deep Learning and Regularization

- 모델 복잡도 이론과 정규화
- 주어진 데이터나 문제에 대해서 가장 최적의 모델 복잡도는 무엇일까(model selection problem)
- 모델 복잡도(model complexity)

데이터나 문제의 복잡도에 비해 모델의 복잡도가 크면, 훈련데이터에 대한 정확도가 우수하나, 과다학습(overfitting)하면 일반화 성능이 저하되는 문제가 발생

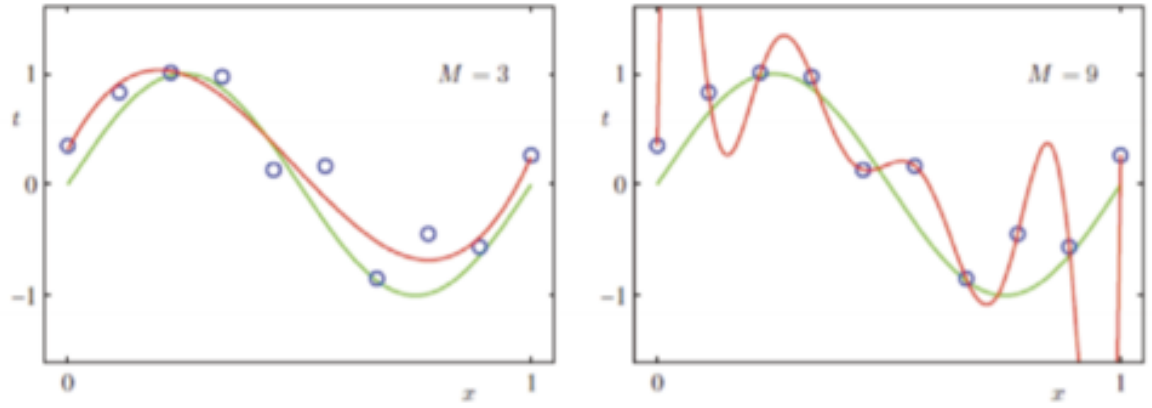


그림 3.9 다항식 신경망의 복잡도에 따른 일반화 성능 비교. 3차 다항식 모델(M=3)에 비해 9차 다항식 모델(M=9)은 과다학습 현상을 보임.

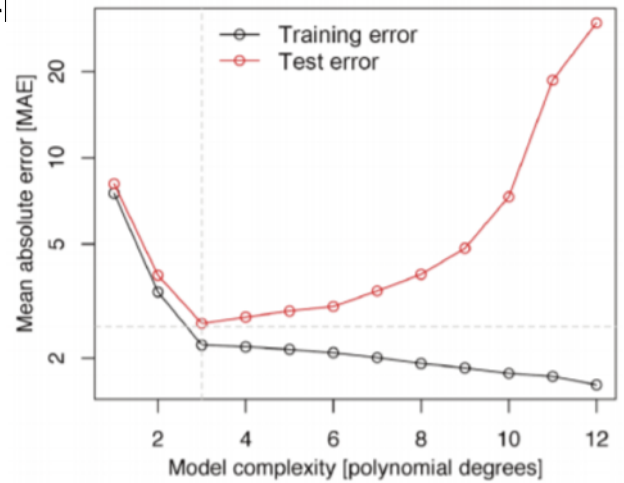


그림 3.10 모델 복잡도와 과다학습 현상의 관계

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Before we start, … Deep Learning and Regularization

- 모델 복잡도(model complexity)
- 일반적으로 모델 복잡도 C(w)는 학습할 매개변수들의 가중치(weight)의 제곱합으로 정의

$$C(\mathbf{w}) = \|w\|^2 = \sum_{k=1}^{K} w_k^2$$

- 가중치의 개수만이 아니라, 가중치의 값이 커져도 모델 복잡도는 증가
- 모델 복잡도를 줄이기 위한 방법
- 가중치 감소법(weight decay)

✓가중치 값을 조절할 때마다, 일정 비율을 감소시키는 방법

- 학습의 반복횟수를 크게 하지 않는것

✓학습 반복횟수가 증가할 수록, 가중치 값들이 커지는 경향을 보임

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Before we start, … Deep Learning and Regularization

- 모델 복잡도(model complexity)
- 목적함수 변경(Objective function 또는 Loss function)

$$F(\mathbf{w}|D) = \beta E(D|\mathbf{w}) + \alpha C(\mathbf{w})$$

- Loss function이 출력값과 데이터 간의 차이를 의미한다면, Objective function은 우리가 실제로 최소, 최대화시키고 싶은 값을 의미
- 목적함수 F(w|D)는 오차(E(D|w))와 모델 복잡도 C(w)에 적절히 계수를 곱해 구해 낼 수 있음

- 구조 위험 최소화(Structural Risk Minimization, SRM)
- 최소 묘사 길이(Minimum Description Length, MDL)
- 최대사후확률법(Maximum A Posteriori, MAP)

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Discriminative and Generative model

- Disciriminative model(변별 모델)

  데이터들을 분류, class 사이의 decision boundary를 model

  ex) 고양이와 개 그림을 구분하는 문제

  주로 supervised learning을 사용, CNN이 원래는 discriminative model로 잘 사용됨

  SVM(Support Vector Machine)

- Generative model(생성모델)

  학습 데이터의 실제 분포를 model

  ex) Fake image를 생성하는 문제

  볼츠만 머신(or RBM), 생성 대립넷(GAN), 오토 인코더

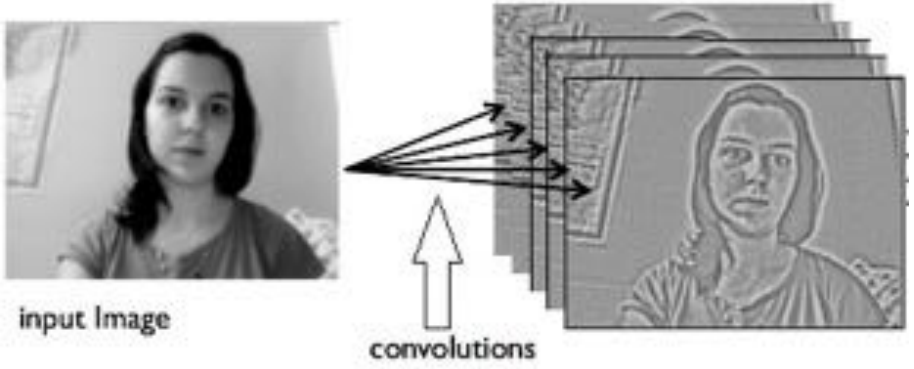
- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

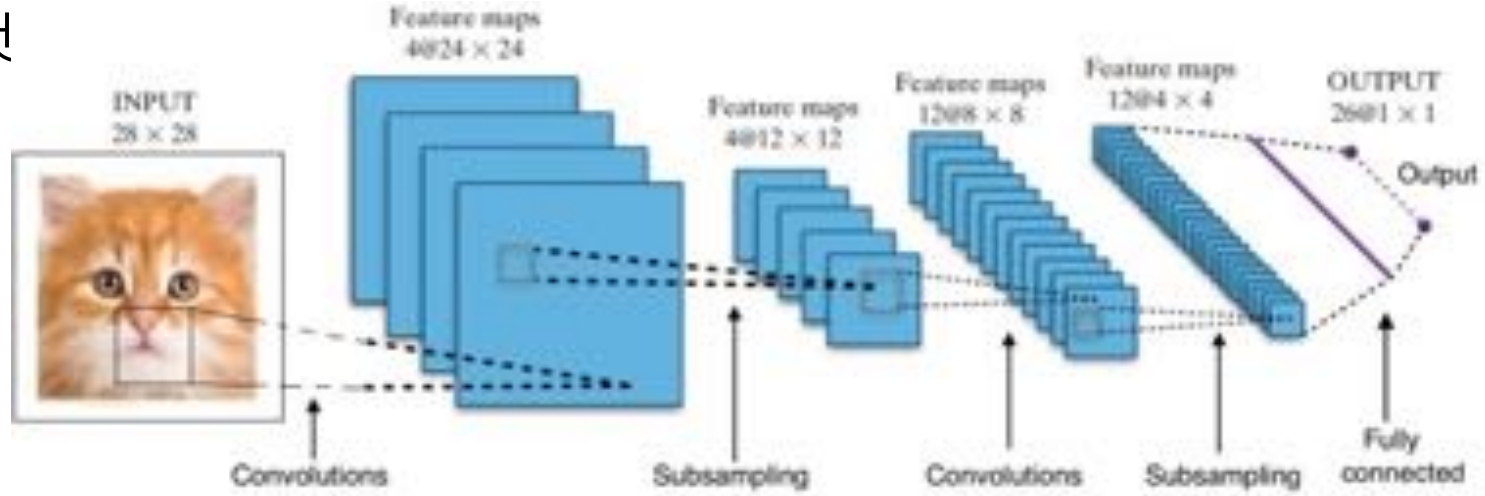
Convolution Neural Network(CNN)

03

- Convolution 신경망

- Convolution 연산을 이용하여
  Feature image를 생성
  다수의 hidden layer를 통과시켜
  context vector를 얻는 것



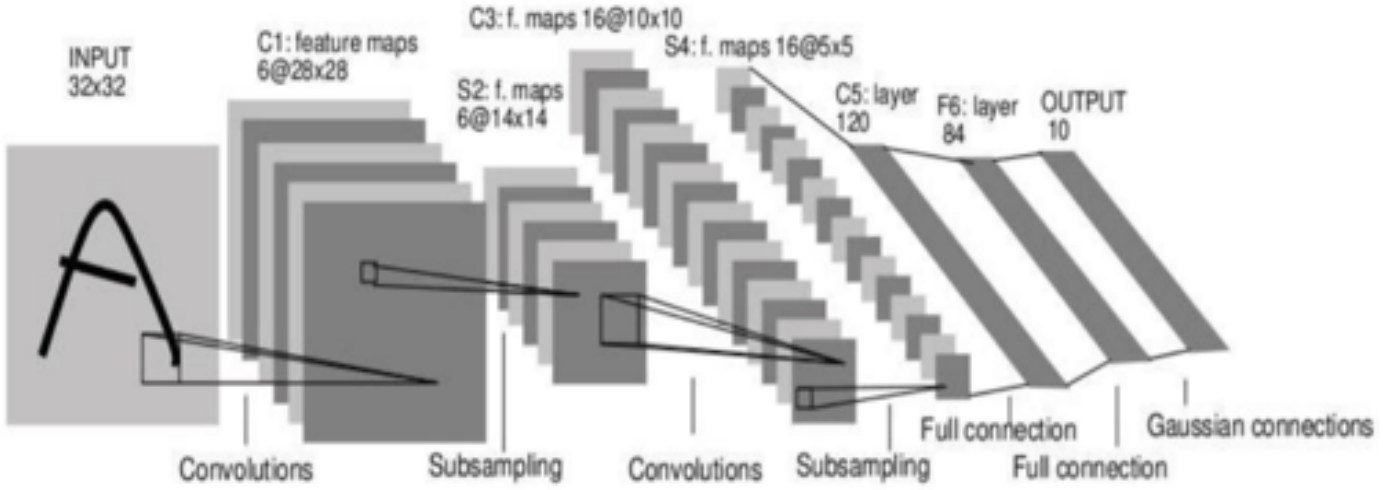


- AI for chemistry KAIST 김우연 교수님 강의 자료 참조

Convolution Neural Network(CNN)

03

- Convolution 신경망
  - 컨볼루션 신경망의 구조
    - 다수의 컨볼루션 층 (C)
    - 다수의 서브샘플링 층 (S)
    - 후반의 완전연결층 (F)



"A"자 문자 이미지를 인식하는 CNN의 구조(LeCon et al., 1998)

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

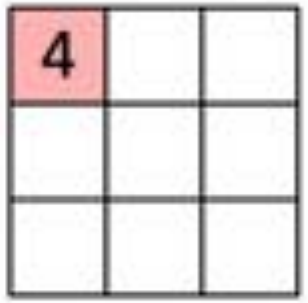
Convolution Neural Network(CNN)

- Convolution 신경망

Convolution Layer

Image

Input: 5×5
Receptive field (or filter): 3×3
Stride: 1
Pooling: summation



Convolved Feature

- AI for chemistry KAIST 김우연 교수님 강의 자료 참조

Convolution Neural Network(CNN)

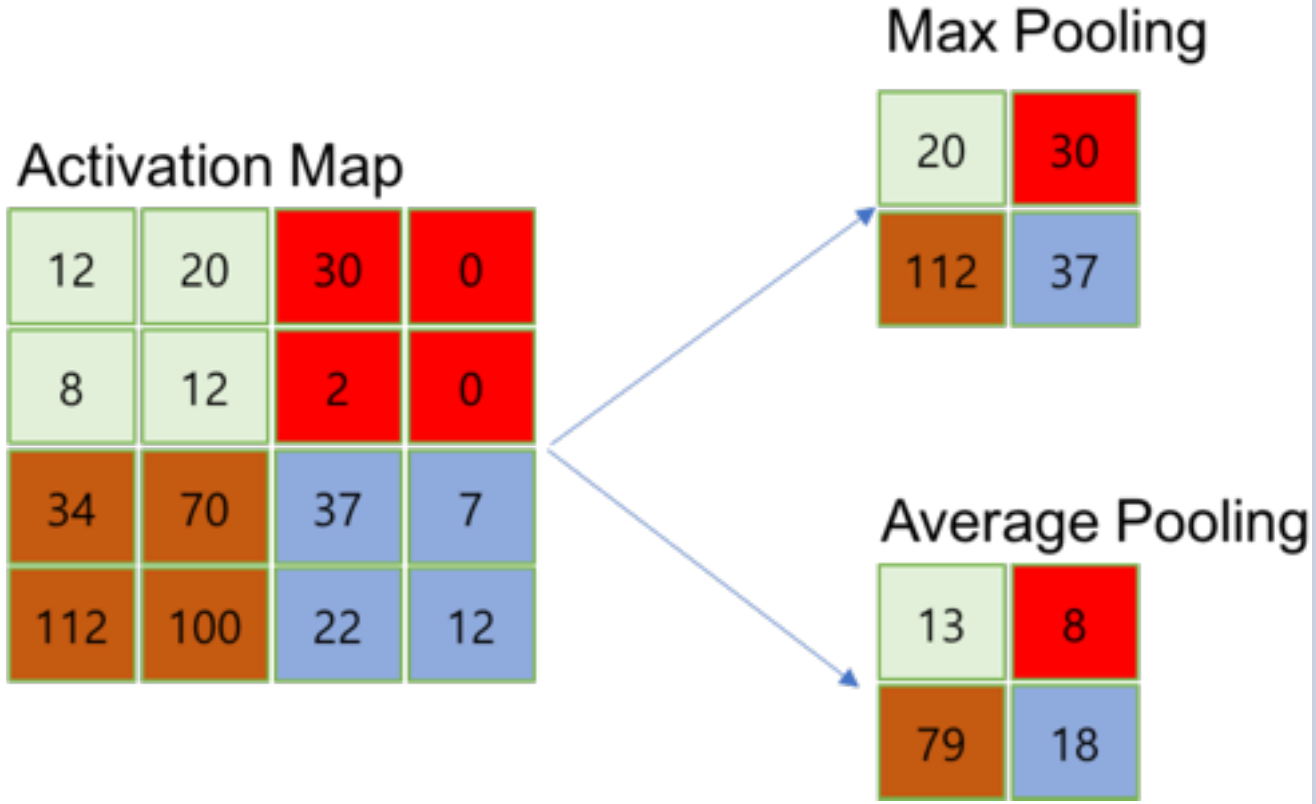
03

- Convolution 신경망

Subsampling Layer

Representation을 작게 만들어 복잡도 감소, Regularization하는 효과

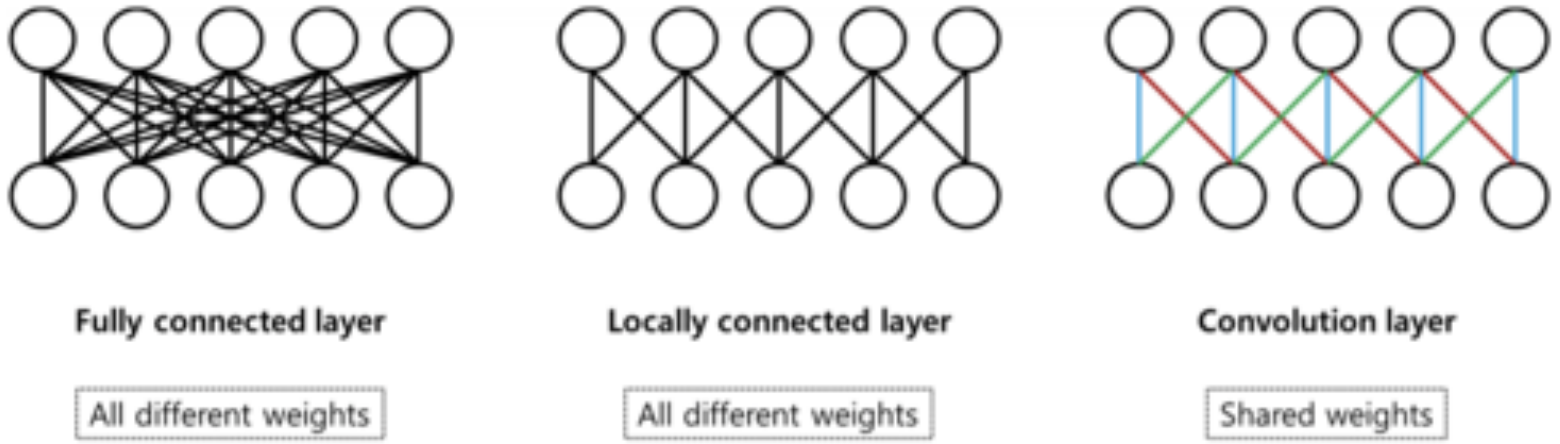
차원 축소만이 아니라, feature map이 이동(shift) 왜곡(distortion)에 강인하게 하는 효과



- AI for chemistry KAIST 김우연 교수님 강의 자료 참조

Convolution Neural Network(CNN)

- Convolution 신경망



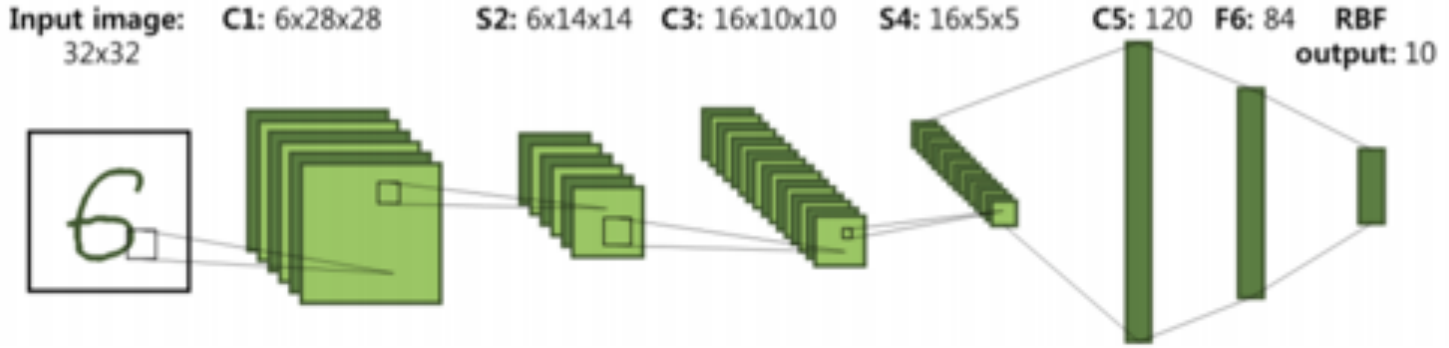
- 연결선(가중치) 종류의 수
  - 완전연결층: 5 x 5 = 25
  - 지역연결층: 2 + 3 + 3 + 2 = 13
  - 컨볼루션층: 3 (같은 색은 같은 가중치)

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Convolution Neural Network(CNN)

CNN의 예: LeNet5

- LeCun에 의해 고안됨(1998)
- 구조
  - 입력(Input) : 32x32 픽셀 이미지
  - 컨볼루션 : 5x5 특징맵 사용
  - 서브샘플링 : ½ 로 축소
  - 완전연결층 : 방사기저함수(RBF) 사용
- 신경망층에 의한 크기 변화
  - 입력 : 32 x 32
  - C1 : 6 x 28 x 28
  - S2 : 6 x 14 x 14
  - C3 : 16 x 10 x 10
  - S4 : 16 x 5 x 5
  - C5 : 120
  - F6 : 84
  - 출력 : 10



- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Convolution Neural Network(CNN)

03

- Convolution 신경망

- 이에 대한 자세한 설명은 AI for chemistry CNN 참조

http://seoklab.org/forum/index.php?topic=5449.0

- Keywords: padding, GoogLeNet(inception module), DeepFace

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Convolution Neural Network(CNN)

- Convolution 신경망 – AlexNet
- Deep Learning 신화의 선두 주
- Drop out method 도입

- 9층 딥러닝 구조 (Krizhevsky, Sutskever, & Hinton, 2012)
  - ILSVRC-2012 1위: 상위 5개(top-5 error) 테스트 에러율 15.3% (2위 : 26.2%)
  - 딥러닝이 기존의 컴퓨터 비전 기법의 성능을 월등히 능가한 최초 사례
  - GPU 사용
  - 후속 딥러닝 연구 촉발

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Convolution Neural Network(CNN)

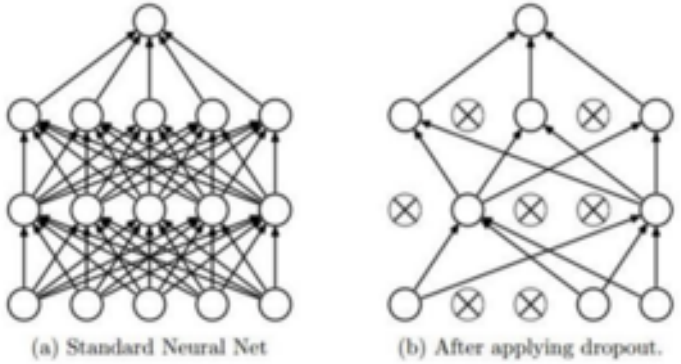
- Convolution 신경망 – AlexNet
- Drop out method 도입(Hinton, et al. 2012)
- Dropout method: 각 은닉 뉴런의 출력을 50%의 확률로 0으로 만듦
- Dropout method as Ensemble Method
- 학습할 때마다 dropout이 각기 다른 뉴런에서 이루어짐
- 이는 여러 architecture를 하나의 architecture로 합치는 방법과 동일
- 즉, AlexNet은 서로 다른 모델들의 예측을 결합하는 방식 – 이미 테스트 에러를 감소 시킬 수 있는 좋은 방법으로 알려져 있었으나 이를 결합하는 것은 현실적으로 어려웠음
- AlexNet의 학습에서 dropout 없이는 overfitting이 발생하였으나 dropout으로 학습시간은 2배정도로 증가하였고, overfitting은 발생하지 않음

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Convolution Neural Network(CNN)

03

- Convolution 신경망 – Dropout method
  - 앙상블 학습(ensemble learning)
    - 서로 다른 모델들의 예측을 결합함으로 테스트 에러를 낮춤
    - 딥러닝 모델은 학습 요구시간이 크기 때문에 앙상블 학습이 어려움
  - 드랍아웃(dropout)
    - 뉴런의 출력을 일정 확률로 0으로 결정 : "제거된" 뉴런은 전파와 역전파 모두 참여하지 않음
    - 매번 다른 구조를 샘플링 → 앙상블 학습과 비슷한 효과
    - 공적응(co-adaptation) 제거 효과: 뉴런간의 의존도를 낮춤
    - 테스트 시 (모든 뉴런의 출력 x 0.5): 드랍아웃망들의 예측 분포의 기하 평
  - AlexNet 적용
    - 완전연결망에만 이용
    - w/o 드랍아웃: 과다학습(overfitting) 현상 발생
    - 수렴 시간은 통상의 2배



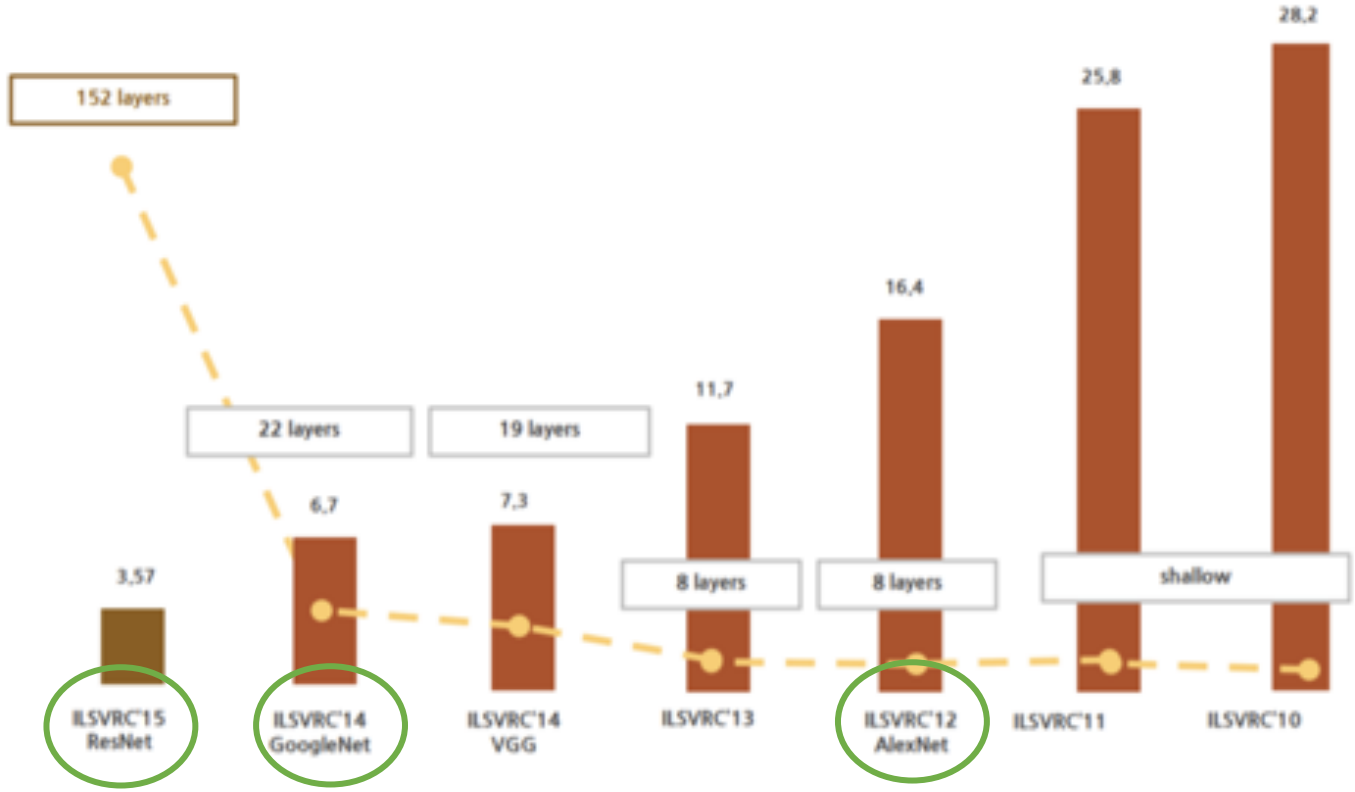
(a) Standard Neural Net

(b) After applying dropout.

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Convolution Neural Network(CNN)

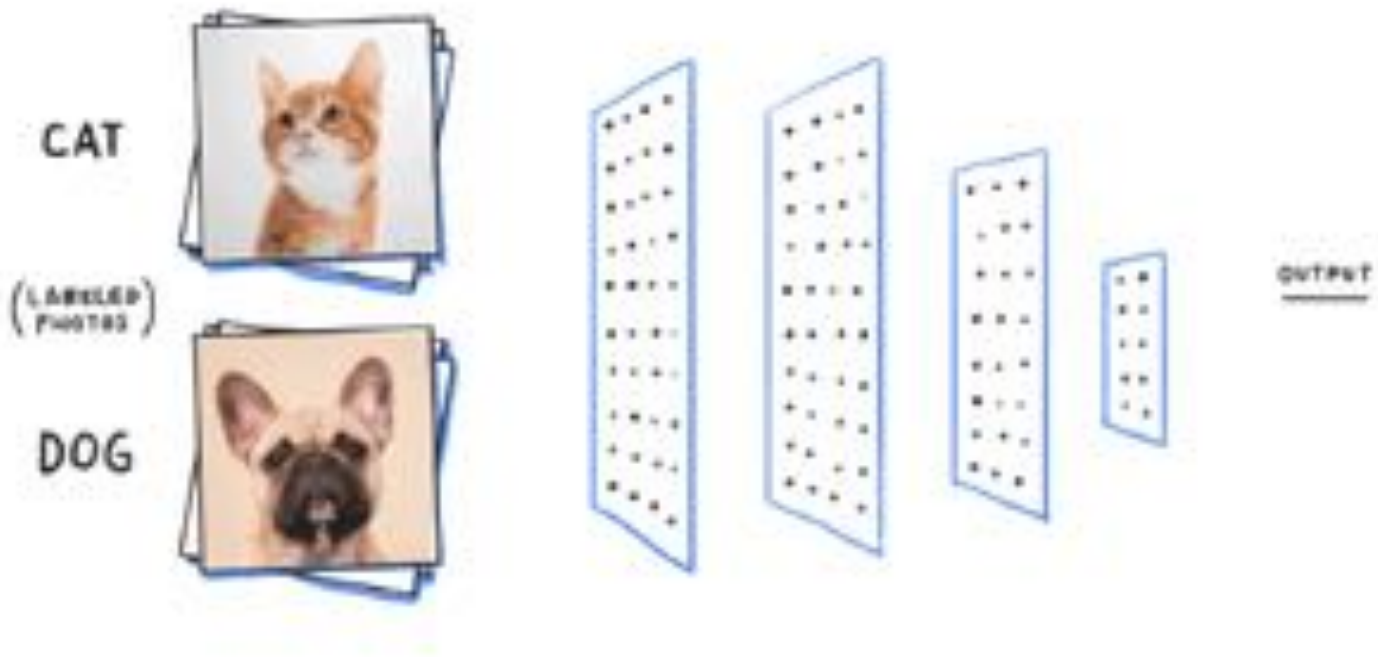
- 이미지 분류의 혁신



- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Convolution Neural Network(CNN)

- Convolution 신경망 정리



- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Recurrent Neural Network(RNN)

- 순환 신경망(Recurrent Neural Network, RNN)
  - 정적인 입출력 사상(static input-output mapping)

$$f_t = f(\mathbf{x}_t; \mathbf{w})$$

    - 출력과 같은 시간의 입력만 고려하는 모델
  - 동적인 입출력 사상(dynamic input-output mapping)

$$f_t = f(\mathbf{x}_t, \mathbf{x}_{t-1}, \ldots, \mathbf{x}_1; \mathbf{w})$$

    - 이전의 모든 시간의 입력을 고려하는 모델
    - **순환신경망**(Rumelhart at el. 1986)이 여기에 해당

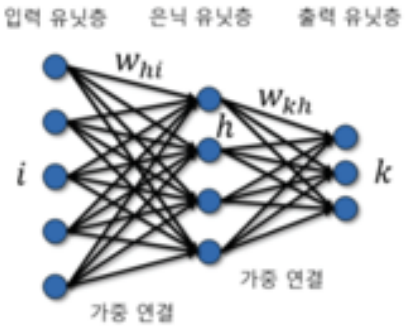


그림 2.8 다층퍼셉트론 신경망(MLP)
정적인 입출력 사상

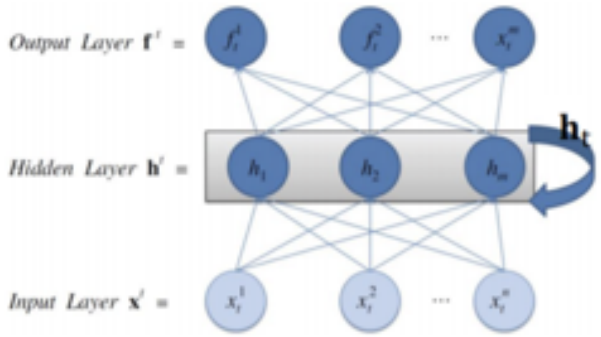
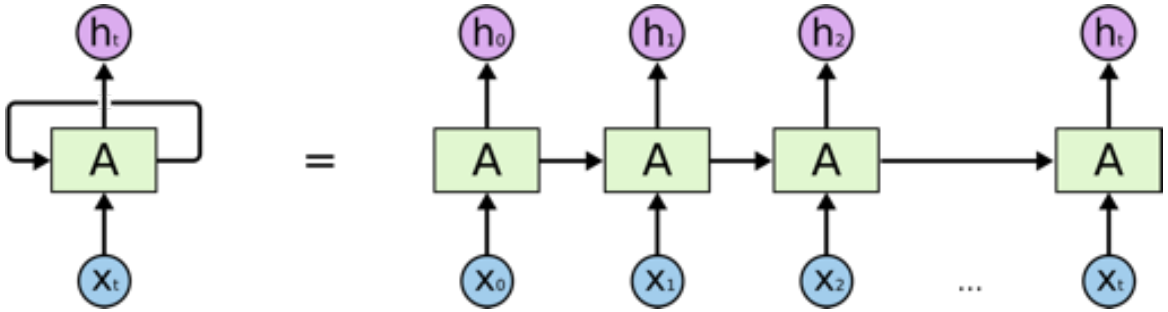


그림 7.2 순환신경망의 시간적 펼침
동적인 입출력 사상

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Recurrent Neural Network(RNN)

- 순환 신경망(Recurrent Neural Network, RNN)



- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

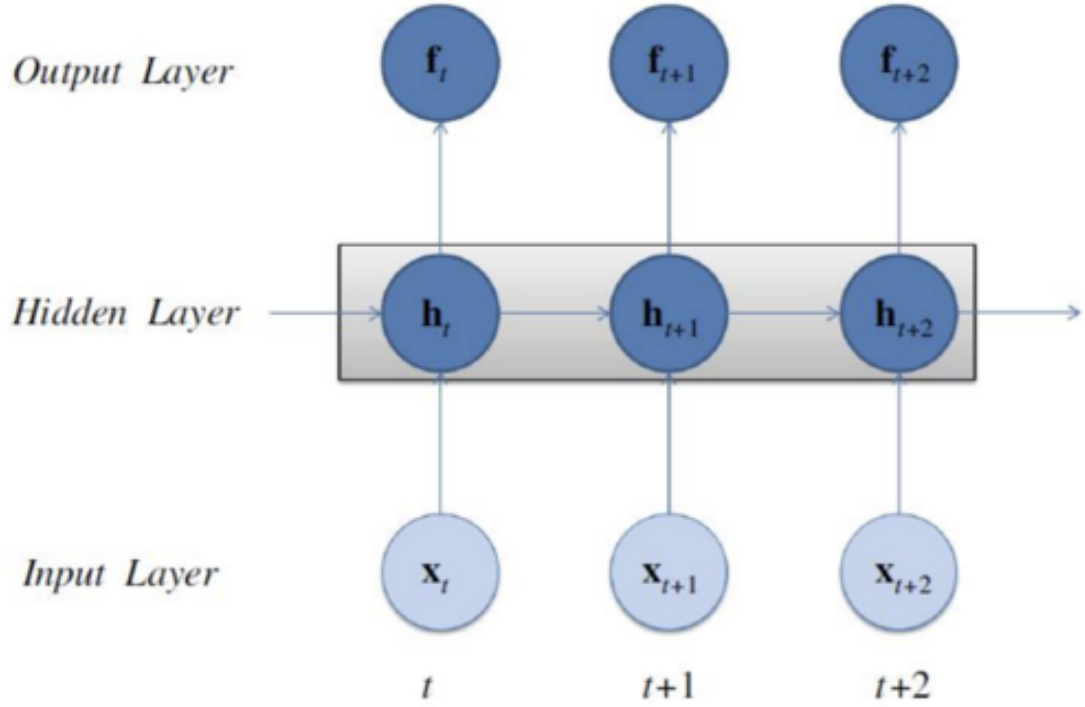
Recurrent Neural Network(RNN)

- 순환 신경망(Recurrent Neural Network, RNN)
- Hidden Layer의 $h_t$state는 이후의 hidden state인 h_{t+1}에 영향

  - 데이터셋: $\{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^{T}$
  - 입력 층: $\mathbf{x}_t$
  - 은닉 층: $\mathbf{h}_t = \phi(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{b})$
  - 출력 층: $\mathbf{f}_t = f(\mathbf{V}\mathbf{h}_t + \mathbf{c})$

U, b 시간 t에 상관없이 고정

t-1의 hidden state에 영향



- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Recurrent Neural Network(RNN)

- 순환 신경망(Recurrent Neural Network, RNN
- RNN은 sequencial data에 유용하게 사용
- Ex) 자연어 번역, SMILES, FASTA 등등
- 다층 구조, 양방향 구조 등으로 다양하게 확장
- 추가적인 설명은 AI for chemistry 자료 참고
- Keywords: BPTT, skip connection

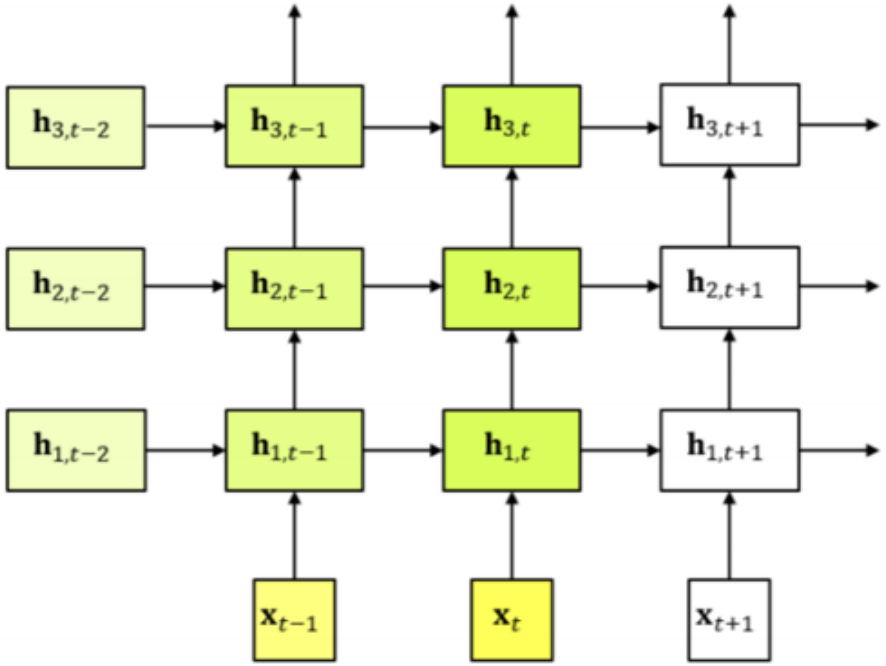


그림 7.3 (a) 다층 순환신경망

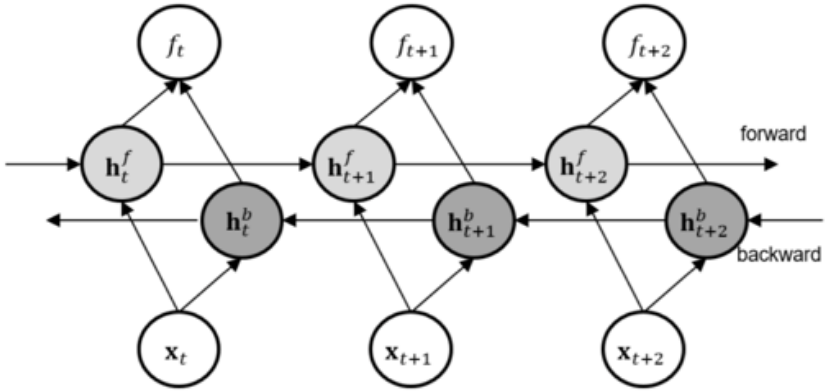


그림 7.3 (a) 잔차(residual) 연결이 추가된 다층 순환신경망

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Recurrent Neural Network(RNN)

- 순환 신경망(Recurrent Neural Network, RNN)
- Gradient vanishing problem: 시간이 지나며 새로운 입력값이 은닉층의 활성도를 덮어쓰면서 민감도가 급격히 감소하고, 결과적으로 망은 최초의 입력값을 잊어버림

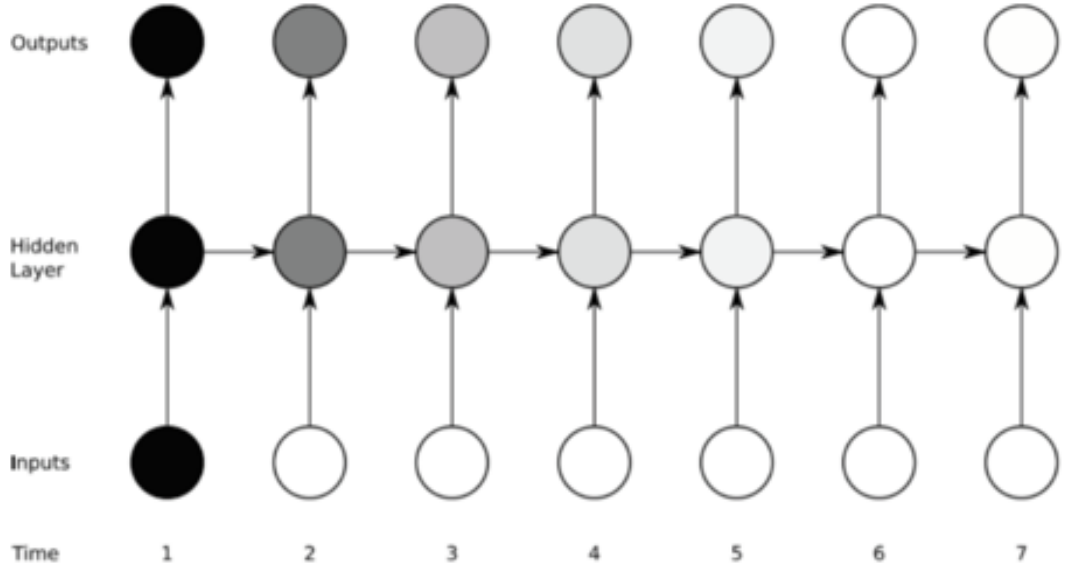


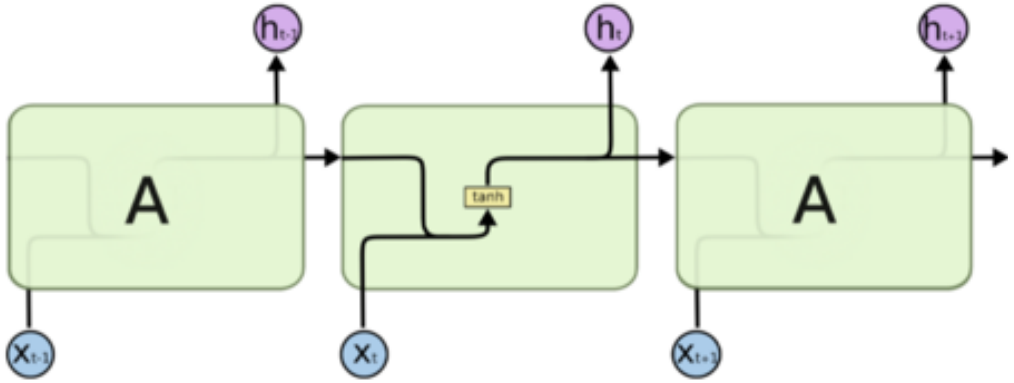
그림 7.6 기본 순환신경망에서 발생하는 오차신호 소실 문제의 도식화(Graves, 2013).

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

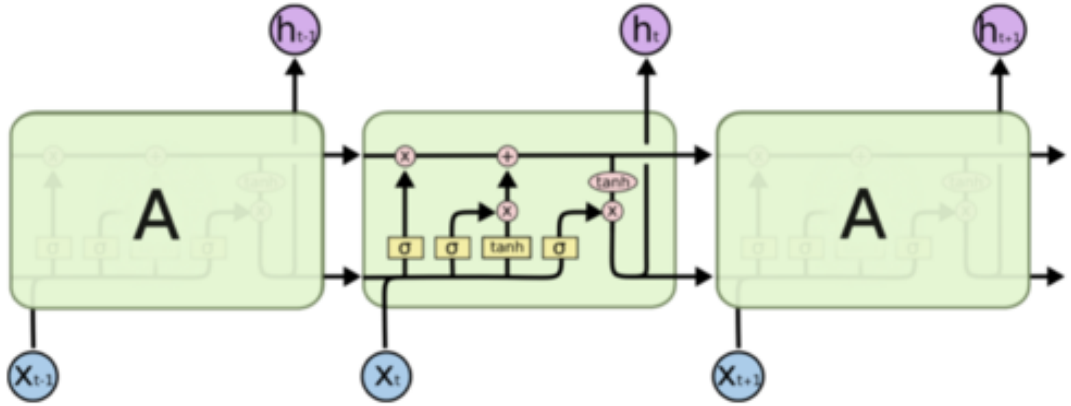
Recurrent Neural Network(RNN)

- LSTM(Long Short-Term Memory, 장단기 메모리)
- 시간에 따른 역전파 에러 신호가 급격히 증가하거나 급격히 감소하는 것을 해결

- 왼쪽: Vanilla RNN, 오른쪽: LSTM



The repeating module in a standard RNN contains a single layer.

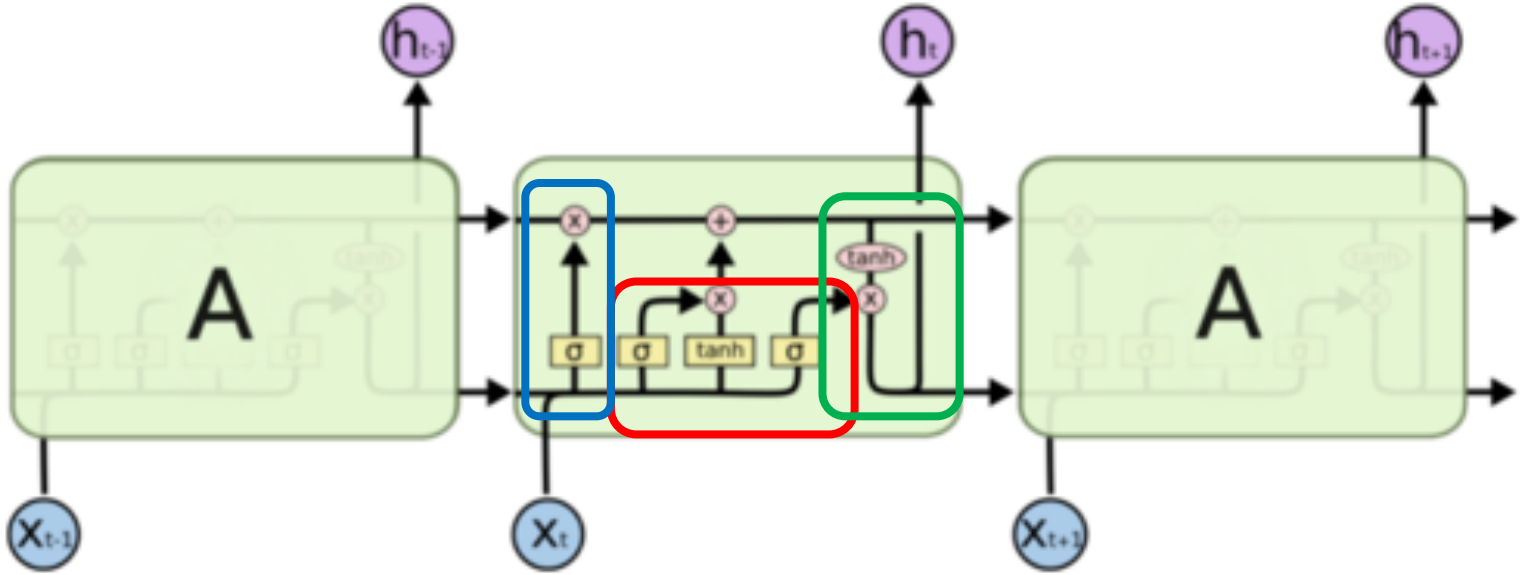


The repeating module in an LSTM contains four interacting layers.

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Recurrent Neural Network(RNN)

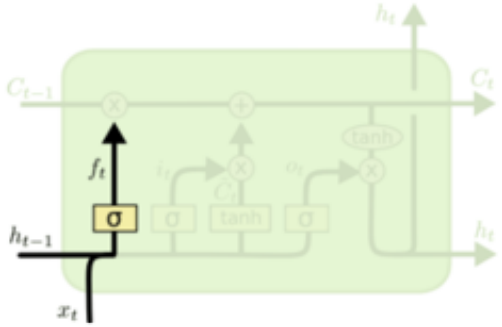
- LSTM(Long Short-Term Memory, 장단기 메모리)
- 장단기 메모리는 이전 hidden state만이 아니라 cell state에 대한 정보도 같이 전달
- 입력



- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

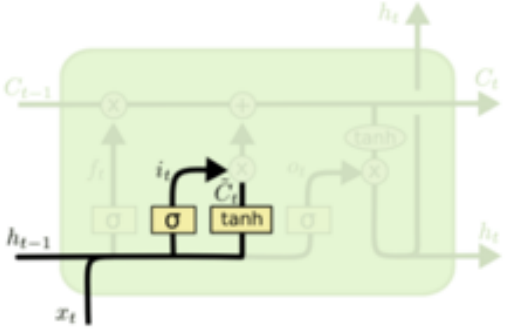
Recurrent Neural Network(RNN)

- LSTM(Long Short-Term Memory, 장단기 메모리)
- 망각(forgot)
- 이전 hidden state h_{t-1}, 입력 x_t
  어느정도 이전 상태를 잊을지
  W_f를 학습, f_t를 출력



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- 입력(input)
  어느 정도 입력을 받을 것인지
  W_i를 학습, i_t를 출력
  임시 Cell state $\tilde{C}_t$를 업데이트



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

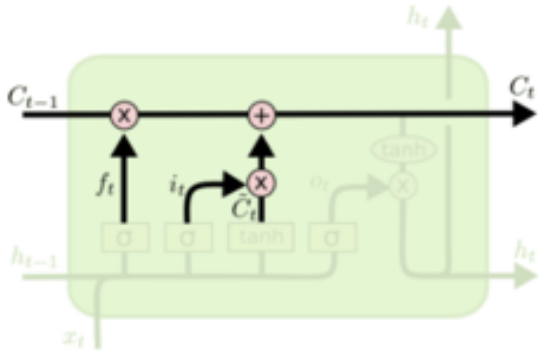
Recurrent Neural Network(RNN)

- LSTM(Long Short-Term Memory, 장단기 메모리)

  임시 cell state $\tilde{C}_t$,

  이전 cell state C_{t-1}로 부터

  이번 cell state C_t 연산

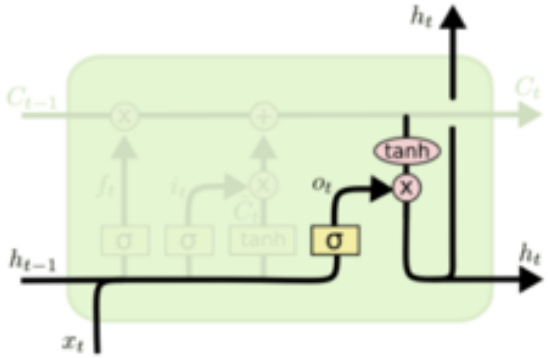


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- 출력(output)

  h_{t-1}, x_t로 부터 output 결정

  h_t를 tanh(C_t)와 o_t의 곱으로 얻어냄

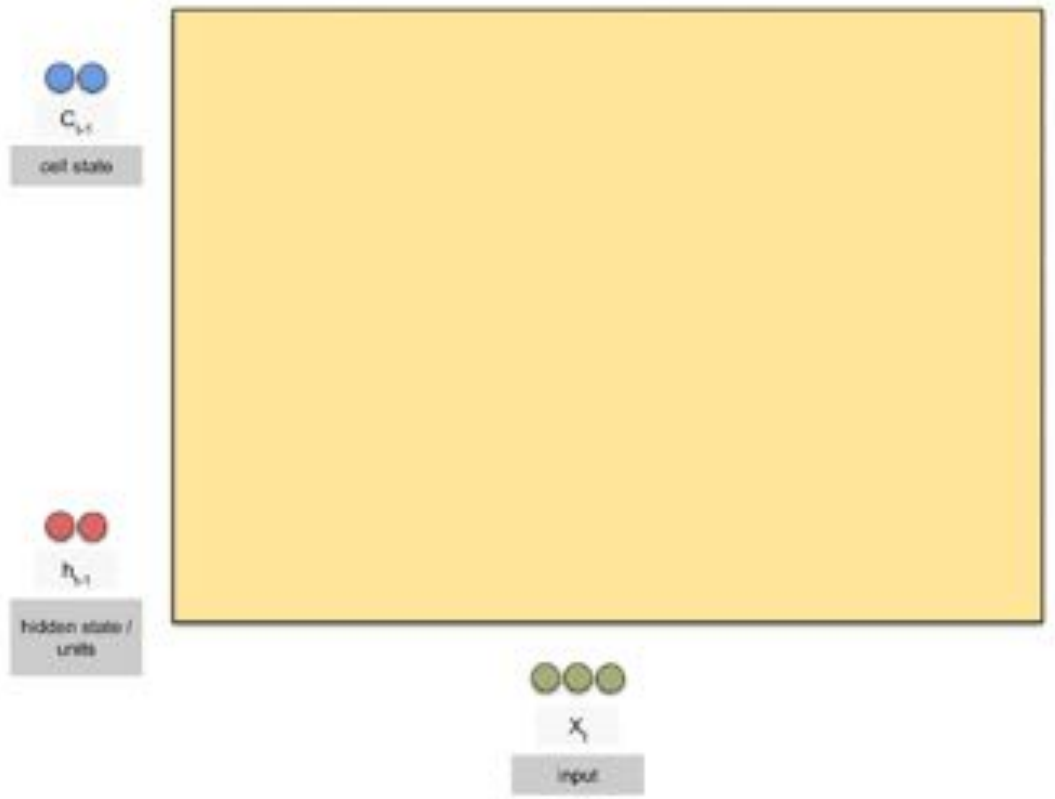


$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Recurrent Neural Network(RNN)

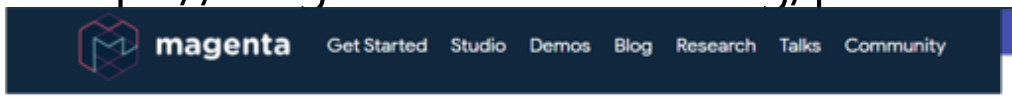
- LSTM(Long Short-Term Memory, 장단기 메모리) 정리



- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

Recurrent Neural Network(RNN)

- LSTM(Long Short-Term Memory, 장단기 메모리) 예시
- https://magenta.tensorflow.org/performance-rnn



Performance RNN: Generating Music with Expressive Timing and Dynamics

Jun 29, 2017
Ian Simon iansimon iansimon
Sageev Oore osageev osageev

We present Performance RNN, an LSTM-based recurrent neural network designed to model polyphonic music with expressive timing and dynamics. Here's an example generated by the model:

0:30 / 0:30

Note that this isn't a performance of an existing piece; the model is also choosing the notes to play, "composing" a performance directly. The performances generated by the model lack the overall coherence that one might expect from a piano composition; in musical jargon, it might sound like the model is "noodling"— playing without a long-term structure. However, to our ears, the *local* characteristics of the performance (i.e. the phrasing within a one or two second time window) are quite expressive.

End-to-End model

- 종단학습(End-to-End Learning)
  - 학습에 필요한 데이터를 거의 원본 그대로 입력하여 중간과정을 명시적으로 모델링하지 않고 직접 출력 데이터 형태로 산출되도록 하는 자동화된 학습 방식
    - 자율주행 자동차(self-driving car)
    - 자연어 질의응답(Q&A) 챗봇
  - 최근 빅컴퓨팅과 빅데이터에 힘입어 딥러닝에 기반한 종단학습으로 해법을 얻을 수 있는 문제의 종류와 복잡도가 크게 증가

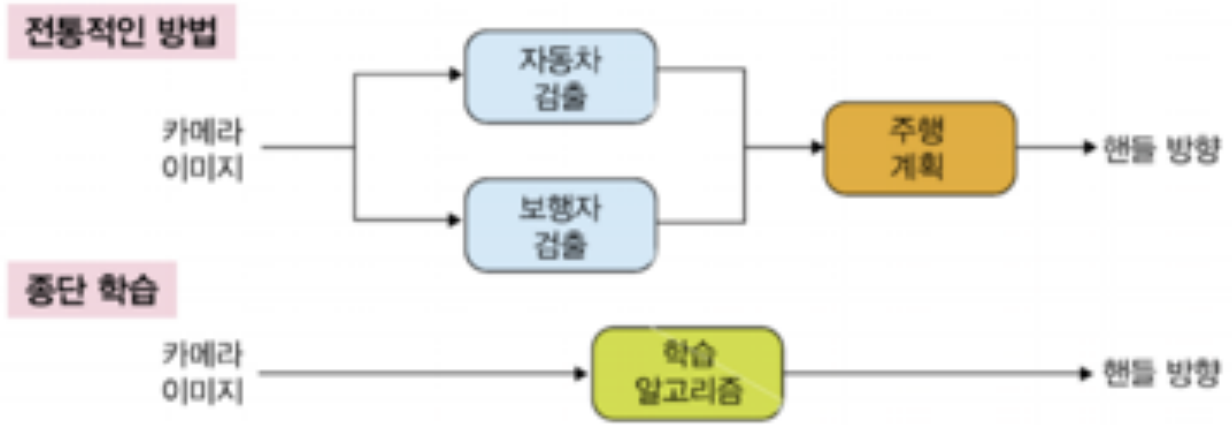


그림 10.1 자율주행에 대한 종단학습의 예시
(Ng, NIPS 2016 Tutorial on Deep Learning for Building AI Systems)

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

End-to-End model

- 종단학습(End-to-End)
- 지금까지 배운 CNN, RNN(LSTM) 등을 이용하여 만드는 모델
- 전통적인 방법보다 성능은 좋으나 더 많은 데이터가 필요
- Ex) 자율 주행 차량에서 자동차와 보행자를 검출하여 회피하는 모델을 만들기 보다는

  운전 시 이미지와 핸들 방향에 대한 데이터를 넣어주고 모델이 핸들 방향을 직접

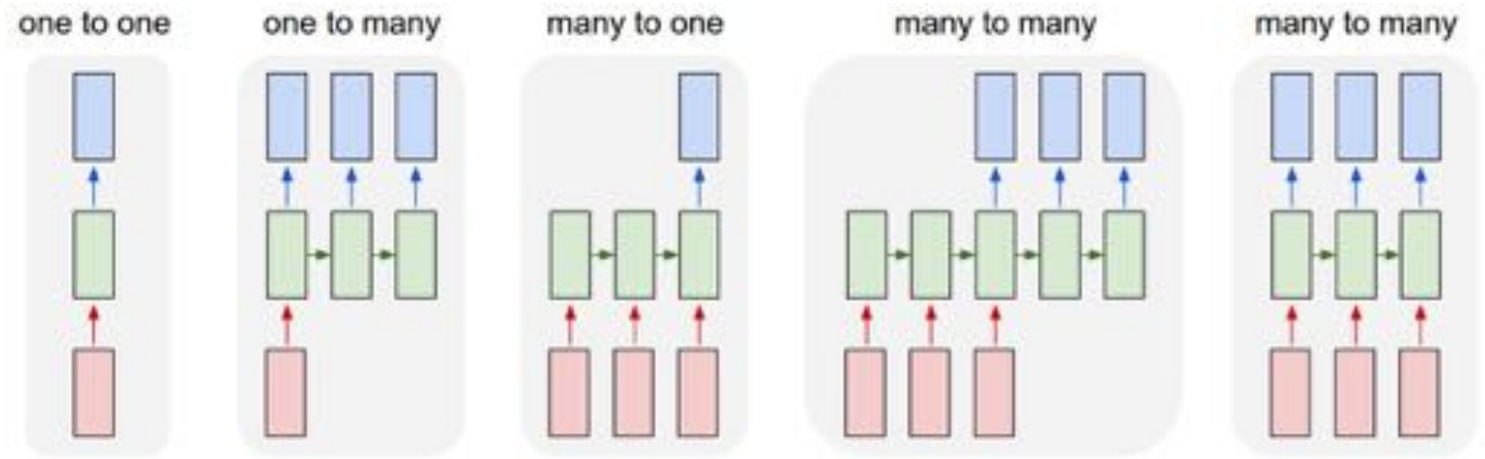
  학습하게 하는 것이 더 유용

- Sequence-to-sequence model(Encoder-Decoder model) (Sutskever et al., 2014, Cho et al., 2014)

- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

End-to-End model

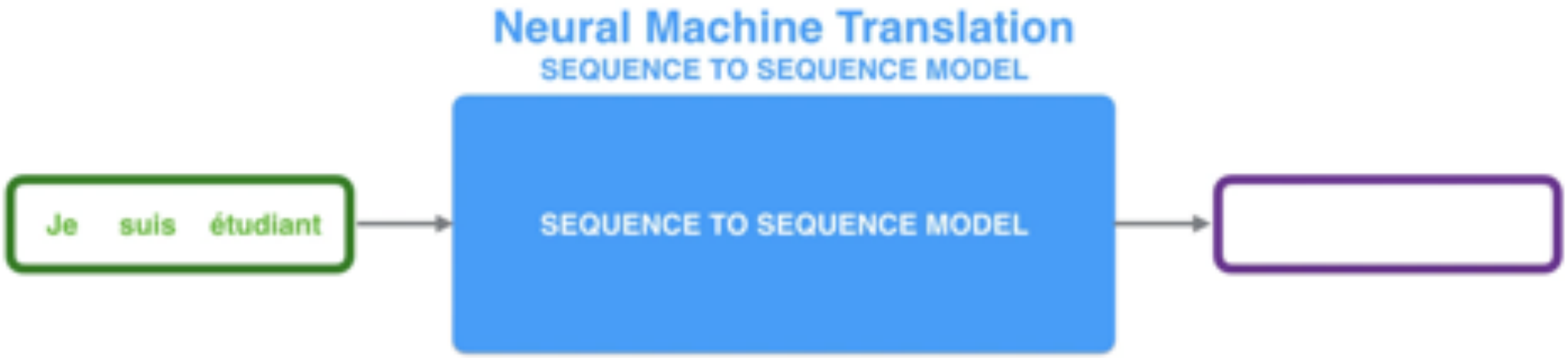
- 예시 - 기계 번역
- LSTM을 사용
- 기계번역은 문장을 넣어 문장을 출력하는 구조
- 문장 안의 단어의 개수가 다수이며, 입출력되는 개수가 동일하지 않으므로 보통 4번째 구조를 사용



- SNU 장병탁 교수님 수업 ppt 및 장교수의 딥러닝 참조

End-to-End model

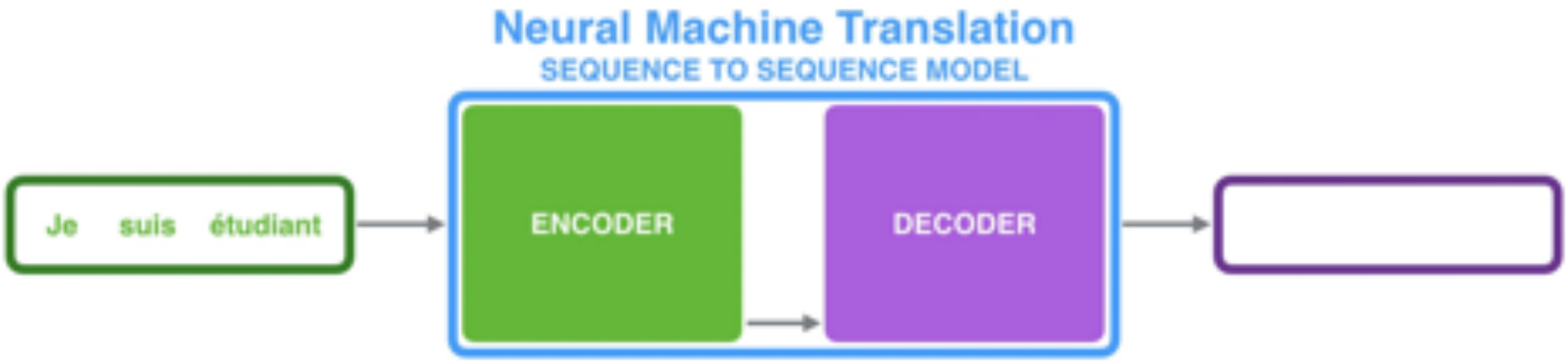
- 예시 - 기계 번역
- Seq2Seq모델로 번역을 하게 된다면..



- https://jalammar.github.io/

End-to-End model

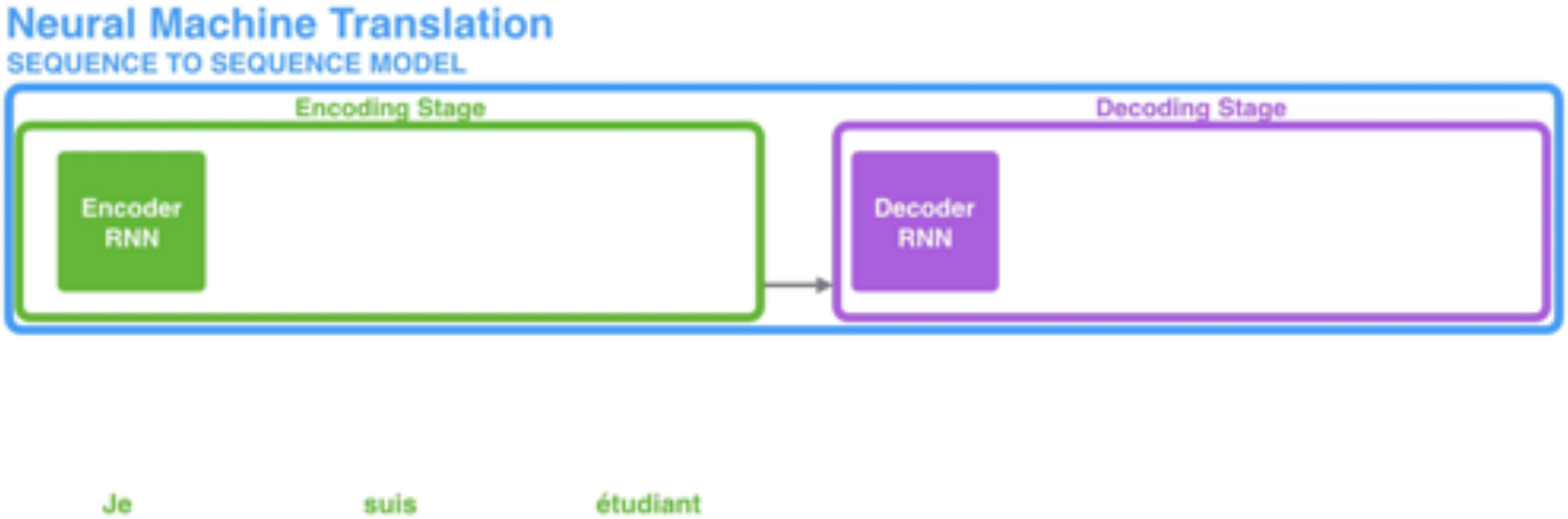
- 예시 - 기계 번역
- Seq2Seq 모델의 내부를 확대해보면..



- https://jalammar.github.io/

End-to-End model

- 예시 - 기계 번역
- Seq2Seq 모델을 RNN(LSTM) level에서 살펴본다면..



- https://jalammar.github.io/

Attention

- Sequencial Data 처리에 있어서 CNN과 RNN의 문제점
- CNN은 Local 영역에 대한 정보를 처리, 가공, 전달
- RNN은 LSTM 역시 long range dependency에 대한 해결 X

  RNN을 이용한 seq2seq은 제한된 길이의 context로 인해 bottleneck 현상이 발생한다는

  문제점이 발생

- Sequencial 데이터를 처리하기 위해 RNN에 보조 개념 Attention이 등장

  어디를 집중해서 관찰할 것인지에 대한 서술

- https://jalammar.github.io/

Attention

- Long range dependency를 해결하기 위한 방법?

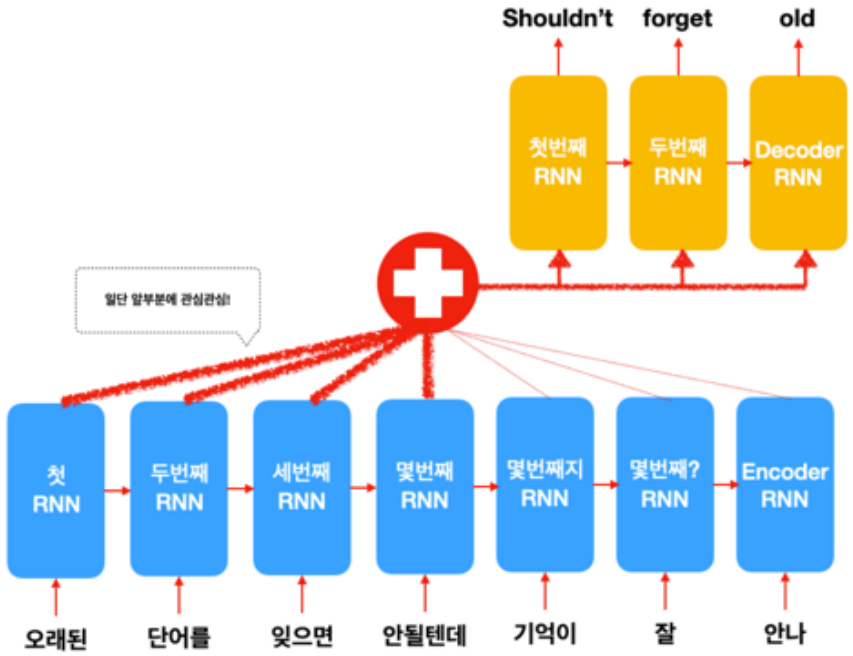
  ➔ 모든 hidden state를 전달해주면 되지 않을까

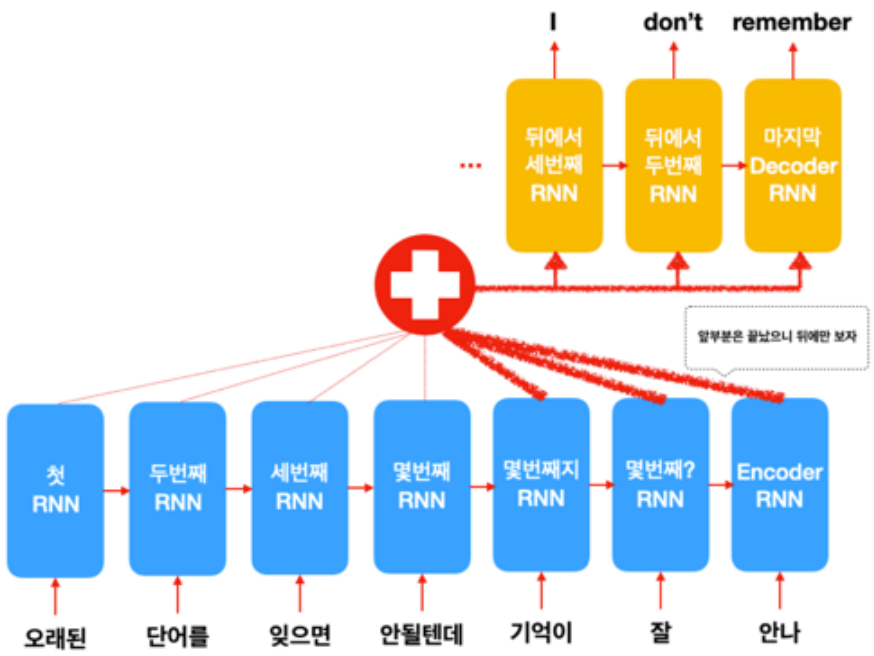
- https://jalammar.github.io/

Attention

- 모든 hidden state를 단순히 전달하게 된다면, Decoder는 인코더로부터 같은 input를 받게 됨 & 메모리 부족

  어떤 hidden state가 어떤 문장에 영향을 더 많이 줄지 결정하자(Attention score)





- https://jalammar.github.io/

Attention

- Attention

- https://jalammar.github.io/

Attention

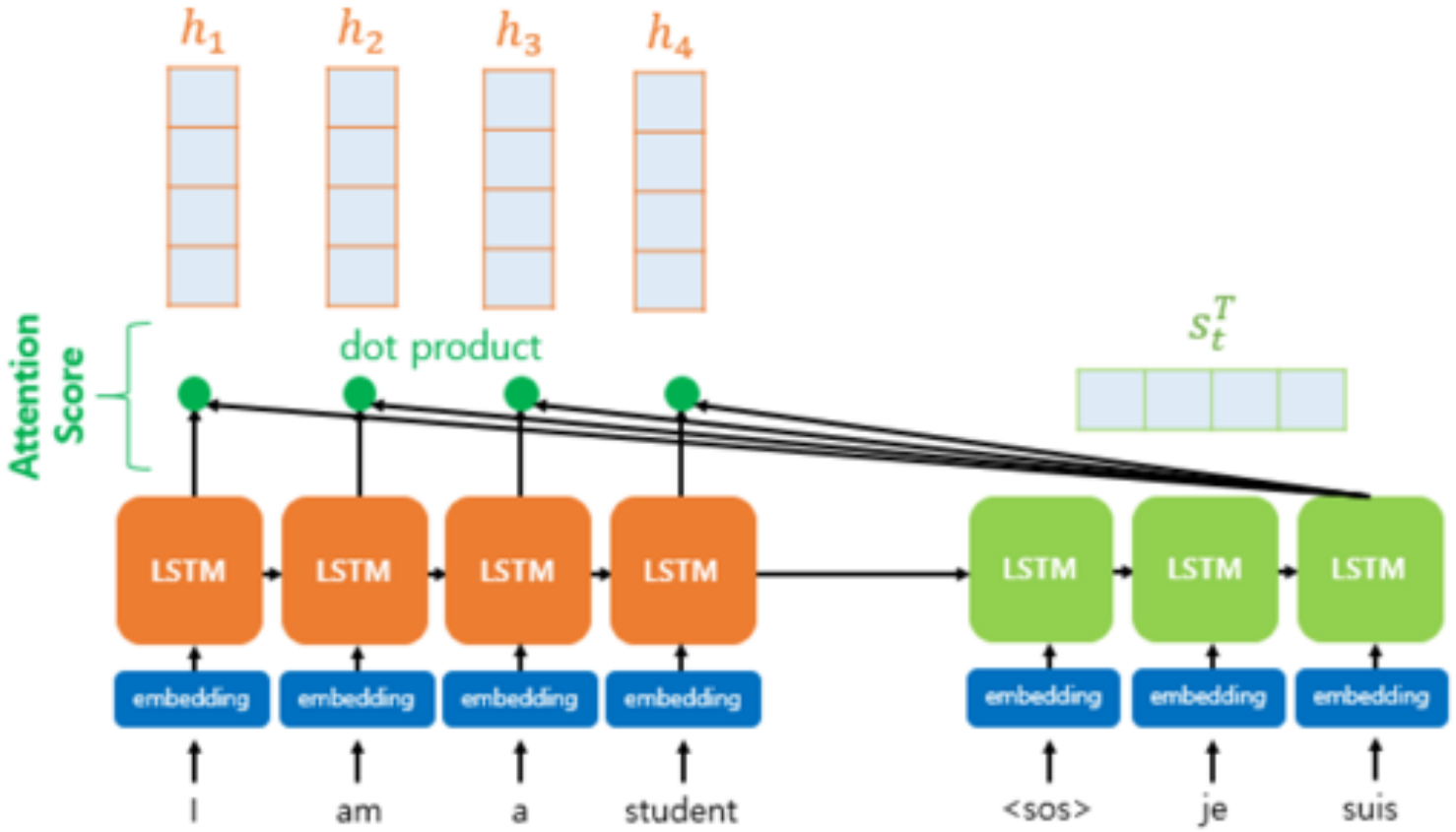
- Attention score를 구하기

인코더의 hidden state들이 디코더의 hidden state들과 얼마나 비슷한지 예측

ex) dot product(내적 값이 클 수록 유사)

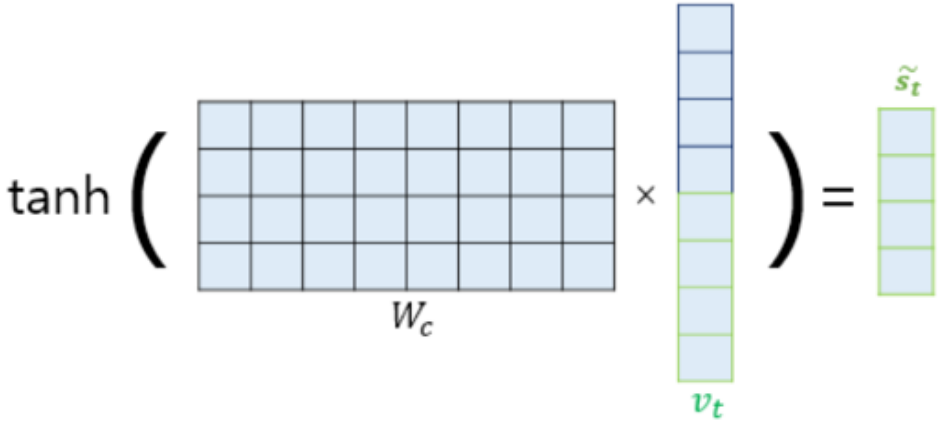
-> softmax에 통과시켜 확률로 변환

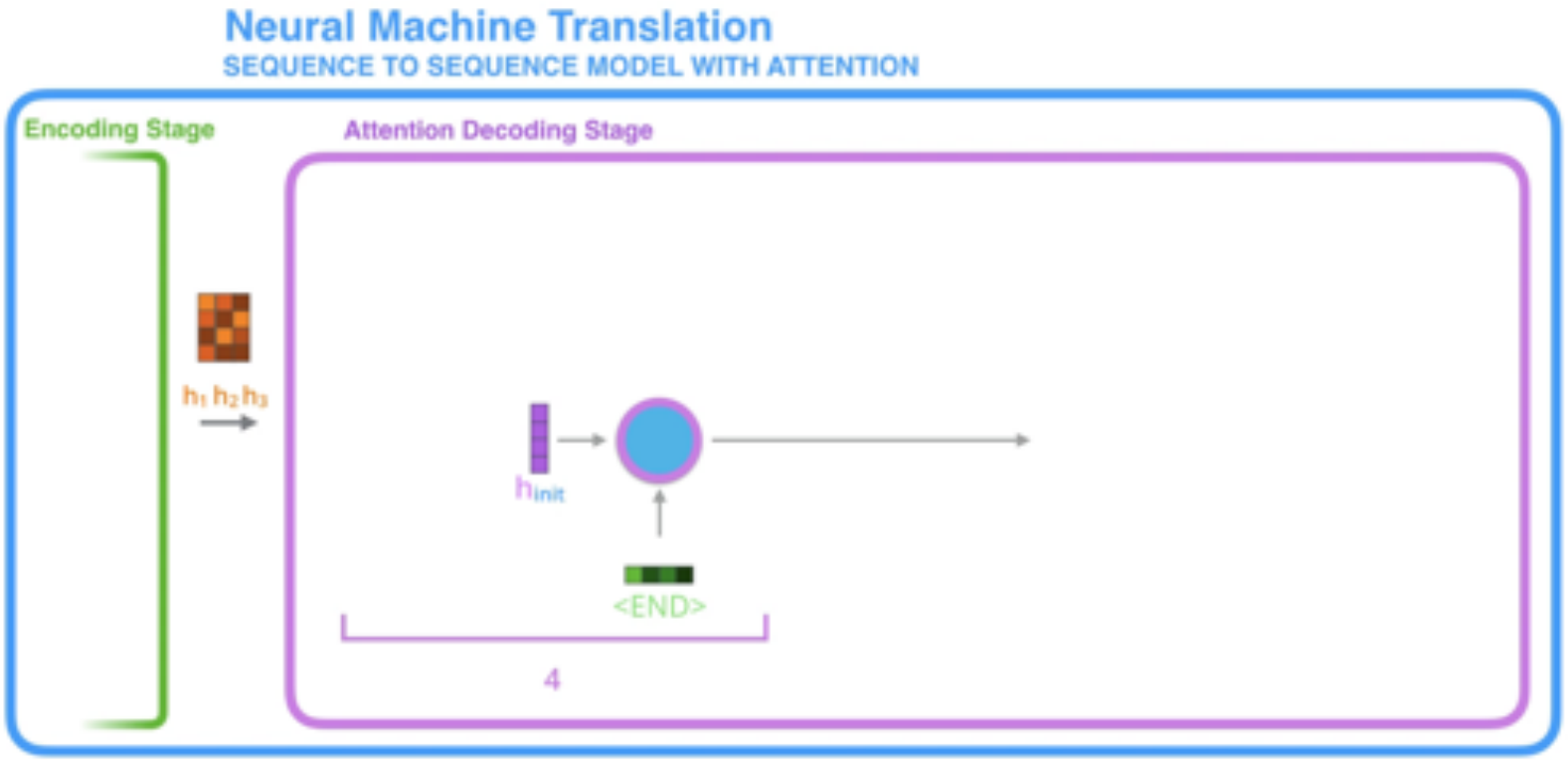
유사도가 높은 hidden state들이 더 많이 결과에 반영하는 방법



- https://jalammar.github.io/

Attention





- https://jalammar.github.io/

Attention

- Attention 정리
- Seq2seq 모델:

  하나의 context vector에 모든 정보를 담아 전달한다면 gradient vanishing 문제를

  해결할 수 없음
- Attention은 모든 hidden-state를 decoder에 전달
- 모든 hidden-state를 decoder에 단순히 전달하기는 메모리의 문제가 발생하므로, 가중합(weighted sum)을 통하여 정보를 전달
- Attention을 이용한 모델에서는 어떤 hidden-state의 weight를 증가시킬 것인지,

  즉, 어떠한 부분의 정보를 더 많이 전달할 것인지 weight를 학습시키는 것을 목표로 함
- https://distill.pub/2016/augmented-rnns/

- https://jalammar.github.io/

Self-Attention & Transformer

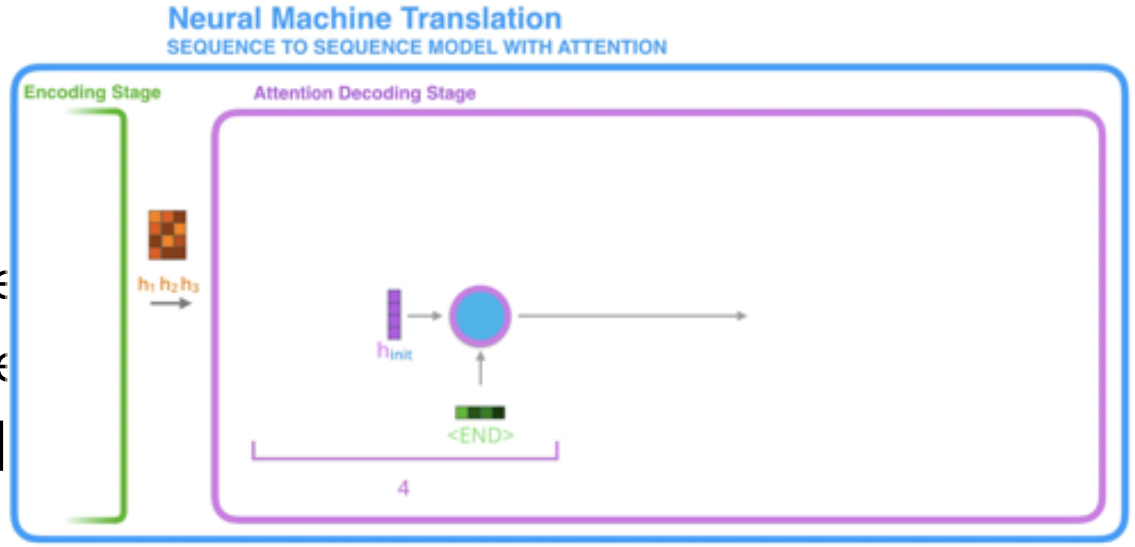
- Self-Attention
- Query, Key, Value (이하 Q, K, V) 가 모두 동일한 attention

- https://jalammar.github.io/

Self-Attention & Transformer

- Attention에서의 표기
- Q(Query), K(Key), V(Value)
- 예시
- Q : t 시점의 decoder의 hidden states
- K : 모든 시점의 encoder의 hidden state
- V : 모든 시점의 encoder의 hidden state

즉, K와 V는 빨간색 벡터 Q는 보라색 벡터



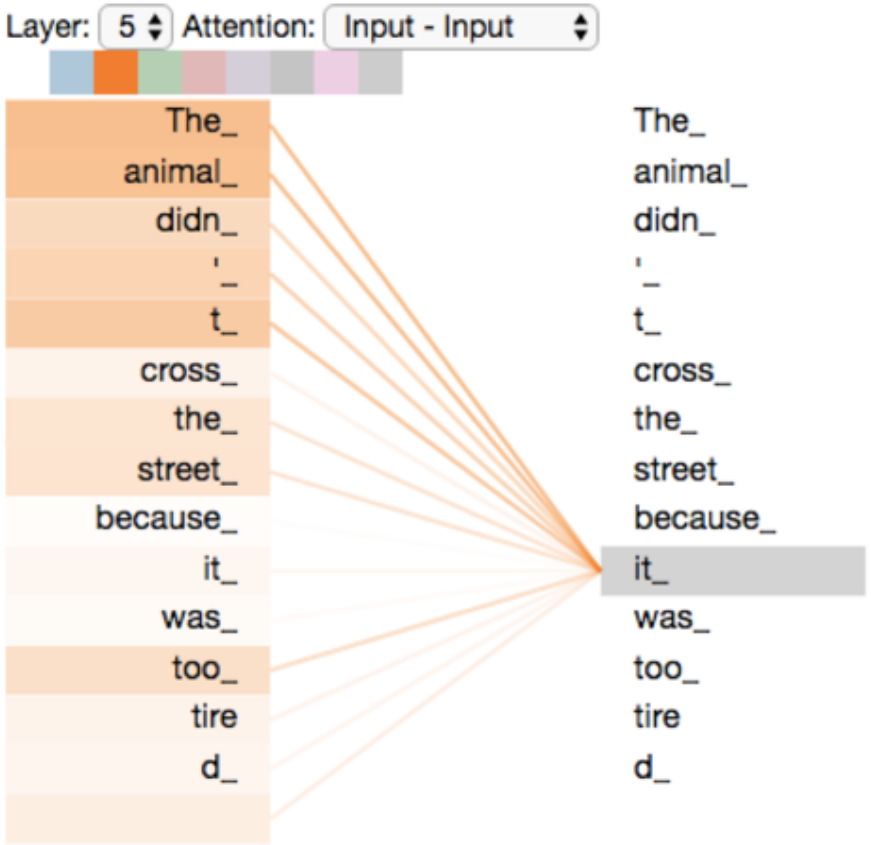
- https://jalammar.github.io/

Self-Attention & Transformer

- Self-Attention
- 예시) self-attention의 query, key, value matrix
- Self attention은 사용하는 Q, K, V의 출처가 같음
- Query와 Key 사이의 유사도를 계산하여 Value를 가중합

Self-Attention & Transformer

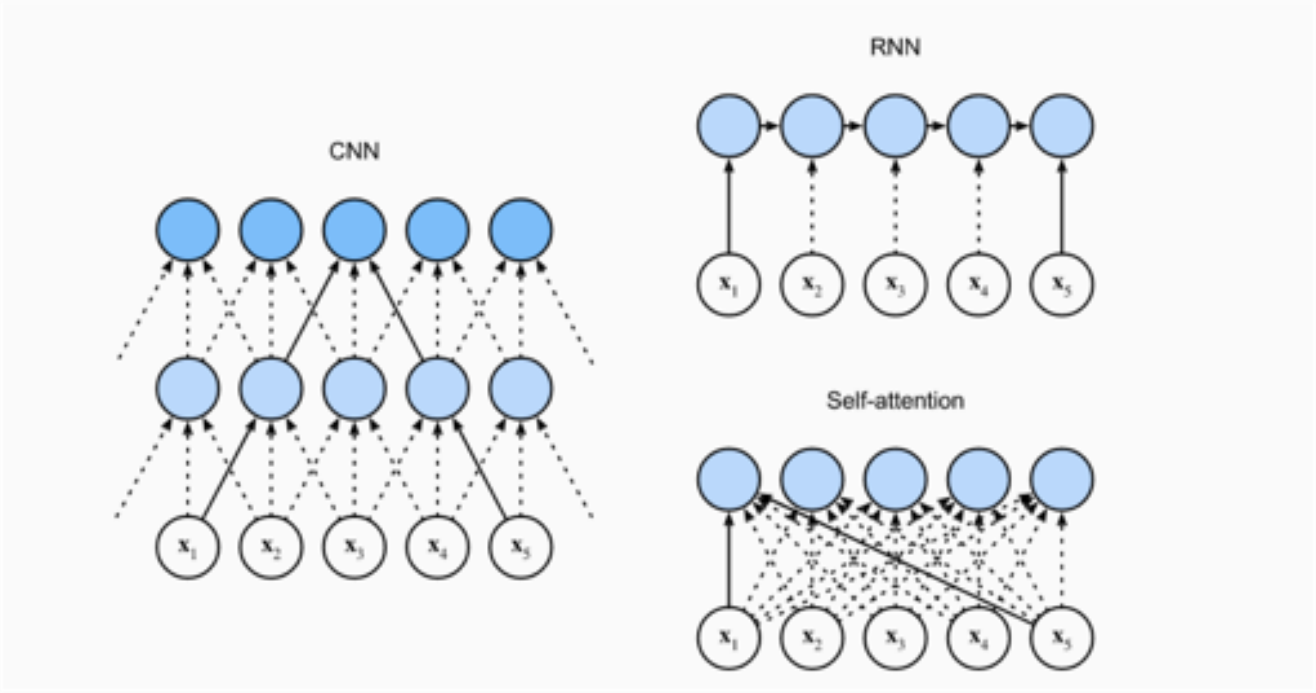
- Self-Attention의 학습 결과 예시



- https://jalammar.github.io/

Attention

- Convolution: a different linear transformation for each relative position
- Self-attention: weighted average



- https://jalammar.github.io/

Self-Attention & Transformer

- Attention is All You Need(2017)
- RNN의 단점을 보조하는 개념으로 사용되었던 attention을
- Attention만을 이용한 Deep Learning Architecture을 개발
- Transformer

RNN with attention

: attention weight를 각각의 sequencial 요소에 곱하여 사용

Transformer

: CNN과 RNN의 long range dependency 문제를 해결

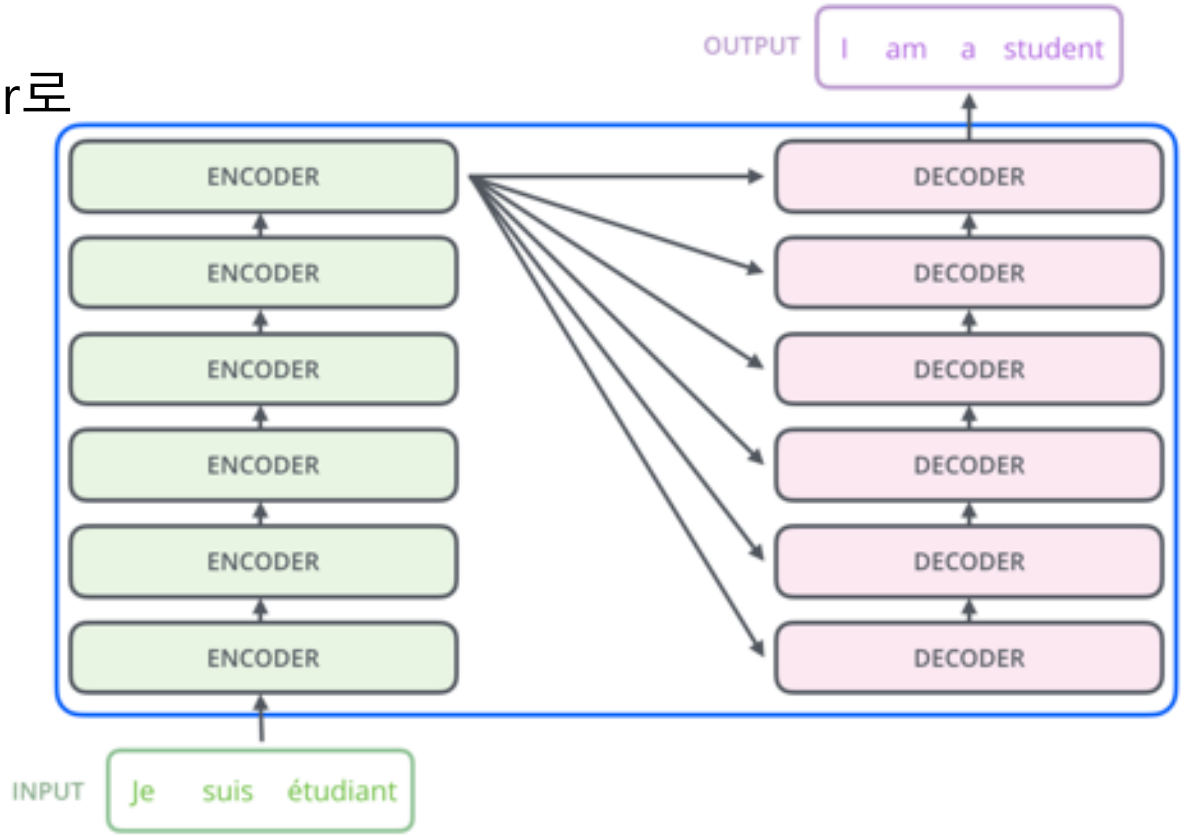
: 인코더-디코더 형식을 유지하면서 self-attention, multi-head attention을 도입하여 sequence 데이터를 병렬처리=> gradient vanishing 문제를 해결

https://nlpinkorean.github.io/illustrated-transformer/

- https://jalammar.github.io/

Self-Attention & Transformer

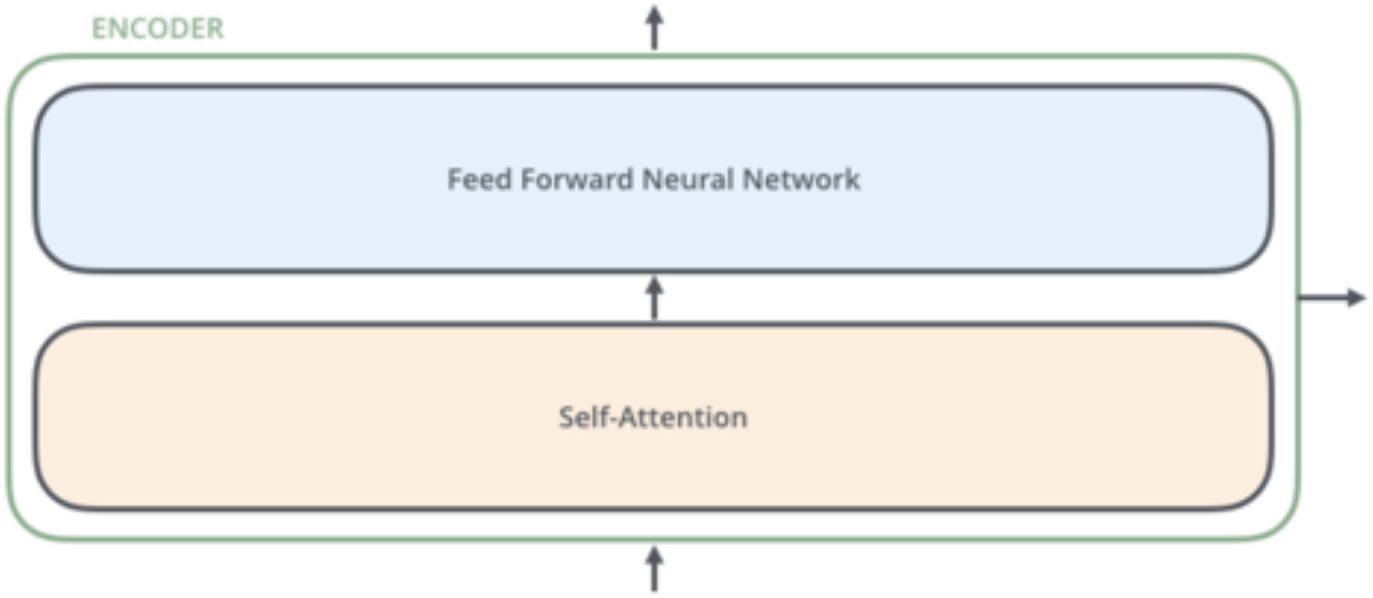
- Transformer
- 6개의 encoder와 6개의 decoder로 구성



- https://jalammar.github.io/

Self-Attention & Transformer

- Transformer
- Encoder 내부는 두 개의 sublayer로 구성

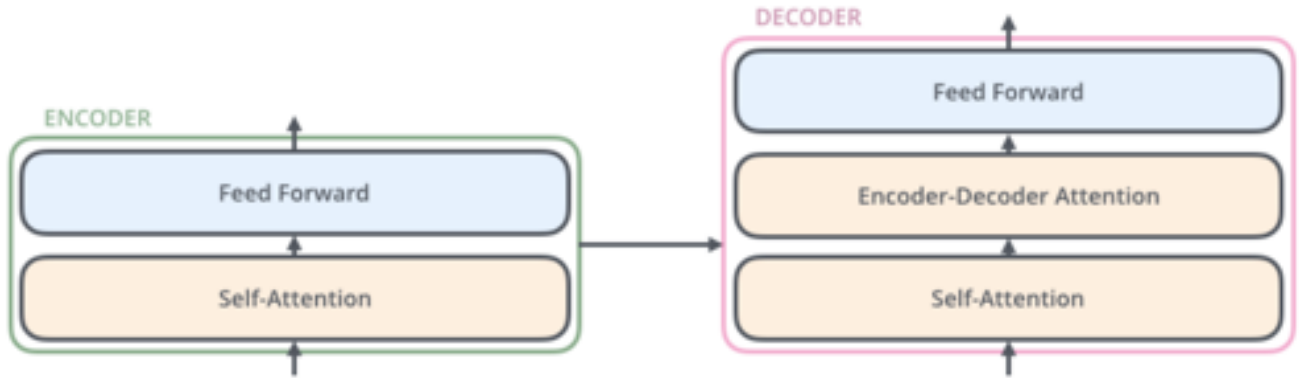


- https://jalammar.github.io/

Self-Attention & Transformer

- Transformer
- Encoder와 Decoder의 구조를 열어보면
- 인코더와 디코더를 연결하는 attention layer
  인코더 내에서 작동하는 self-attention
  디코더 내에서 작동하는 self-attention로

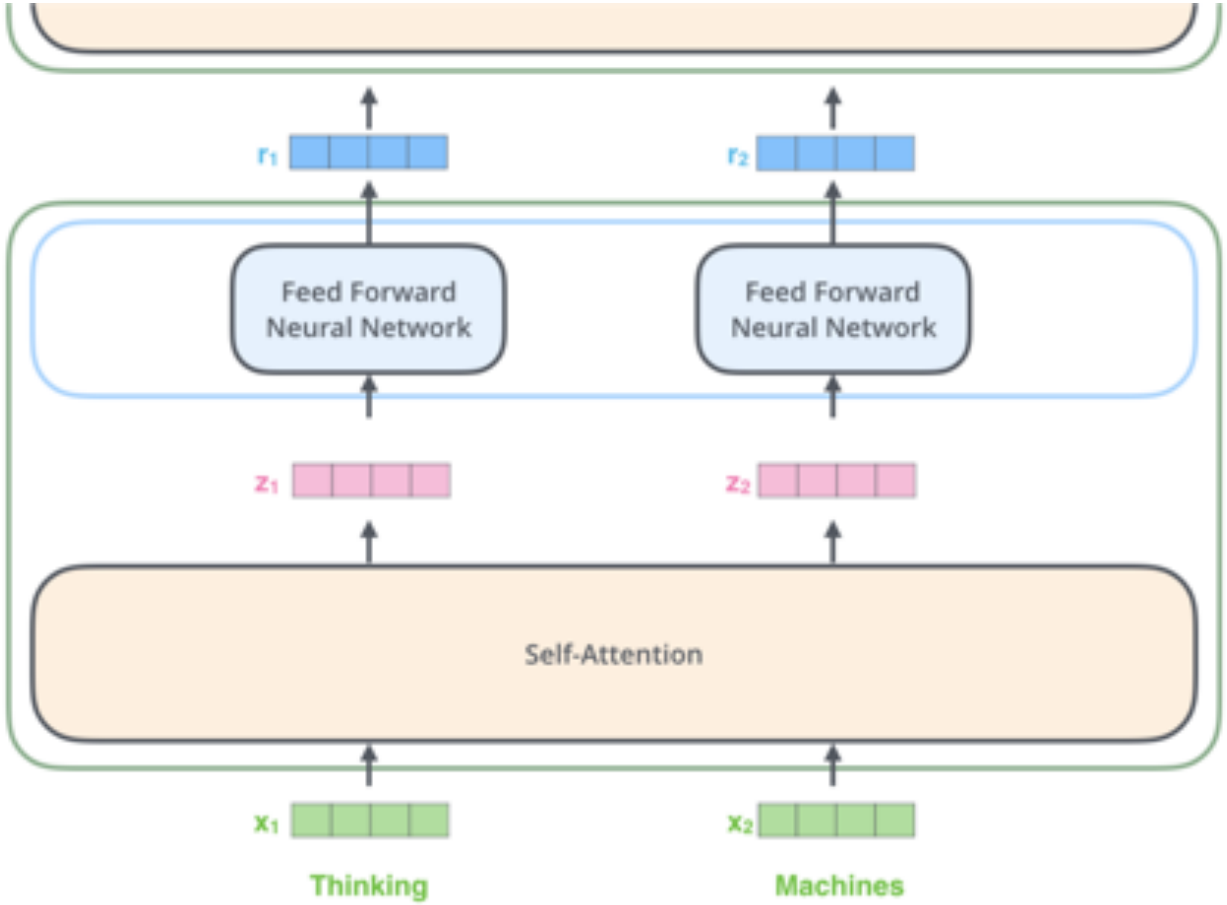
총 3가지의 attention layer가 있음을 확인 가능



- https://jalammar.github.io/

Self-Attention & Transformer

- Transformer
- Encoder의 구조를 열어보면..

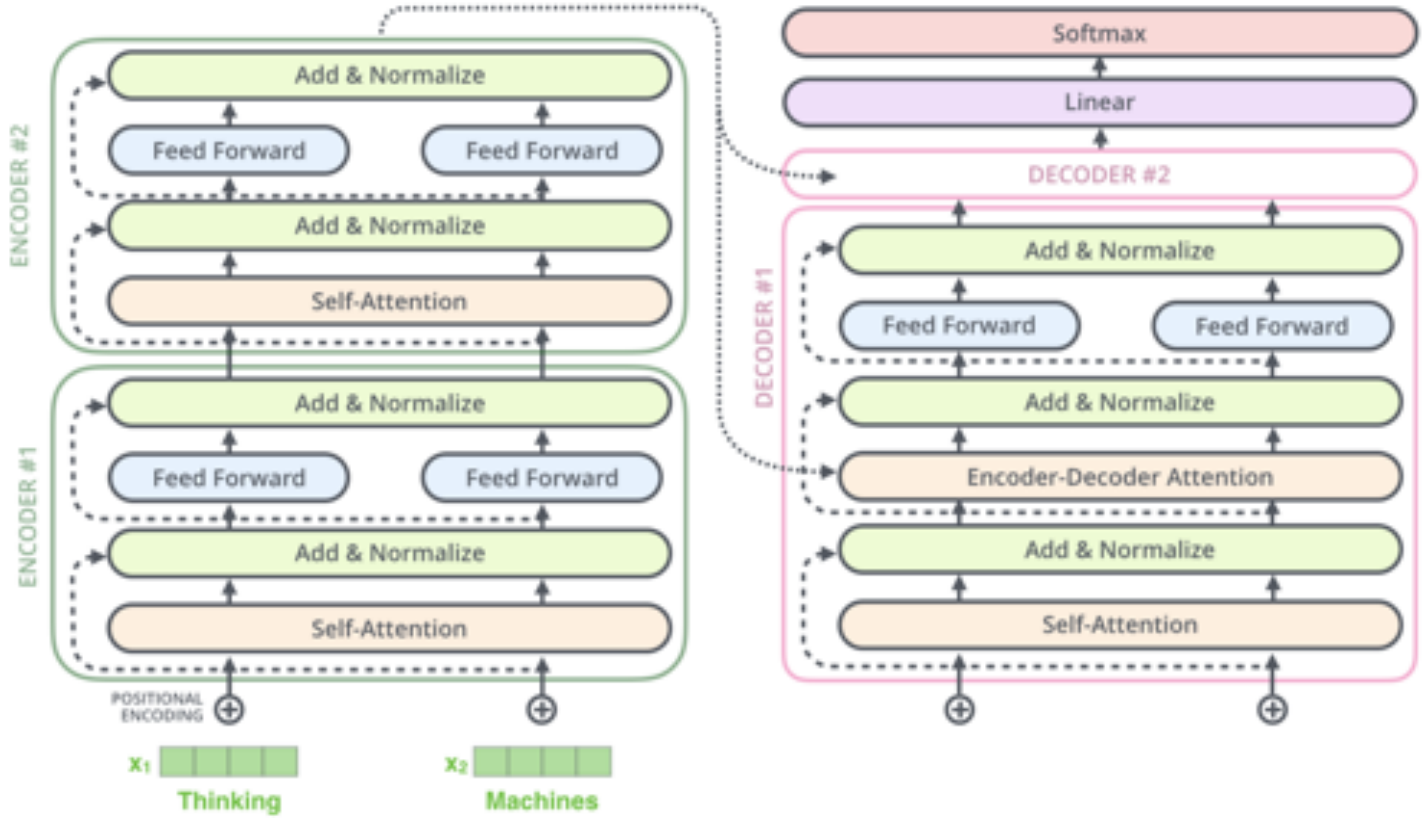


- Self-Attention Layer
- 단순 순방향 신경망

- https://jalammar.github.io/

Self-Attention & Transformer

- Transformer: 간단히 2개의 encoder와 2개의 decoder가 있는 경우를 묘사한다면..



- https://jalammar.github.io/

Self-Attention & Transformer

- Transformer
- Self-Attention
- Self-attention을 하게 되면 decoder vector(Query)와 encoder vector(Key, Value)가 동일한데 둘 사이의 유사도를 계산하는 weight score가 자기 자신에게 제일 크게, (0,1,0,0)과 같이 나타나지는 않게 될까?
- Multi-head attention

모델이 다른 위치에 집중하는데 도움

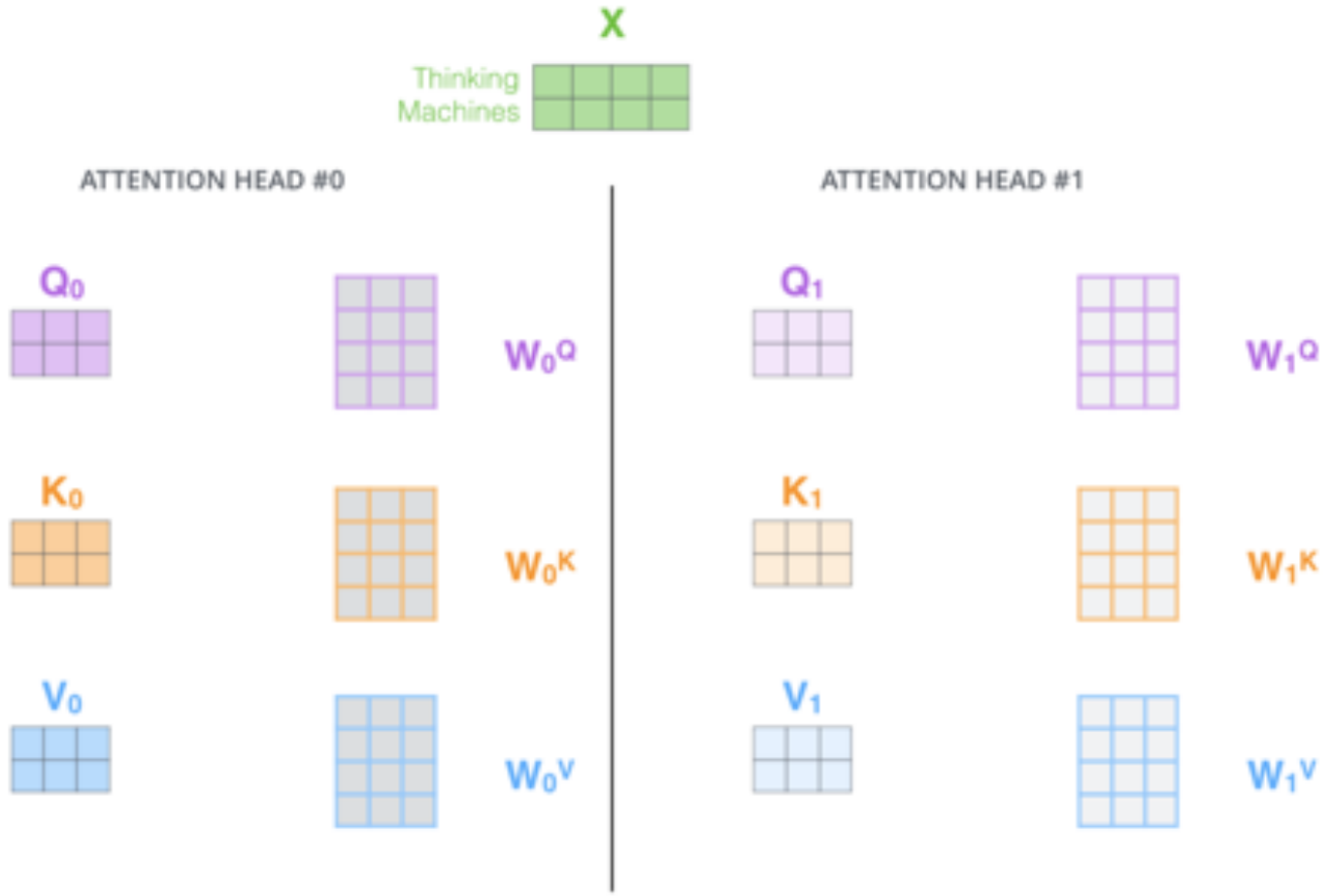
Attention layer가 여러 개의 representation 공간을 가지게 도움

(다양한 Q, K, V가 만들어 지기 때문)

- https://jalammar.github.io/

Self-Attention & Transformer

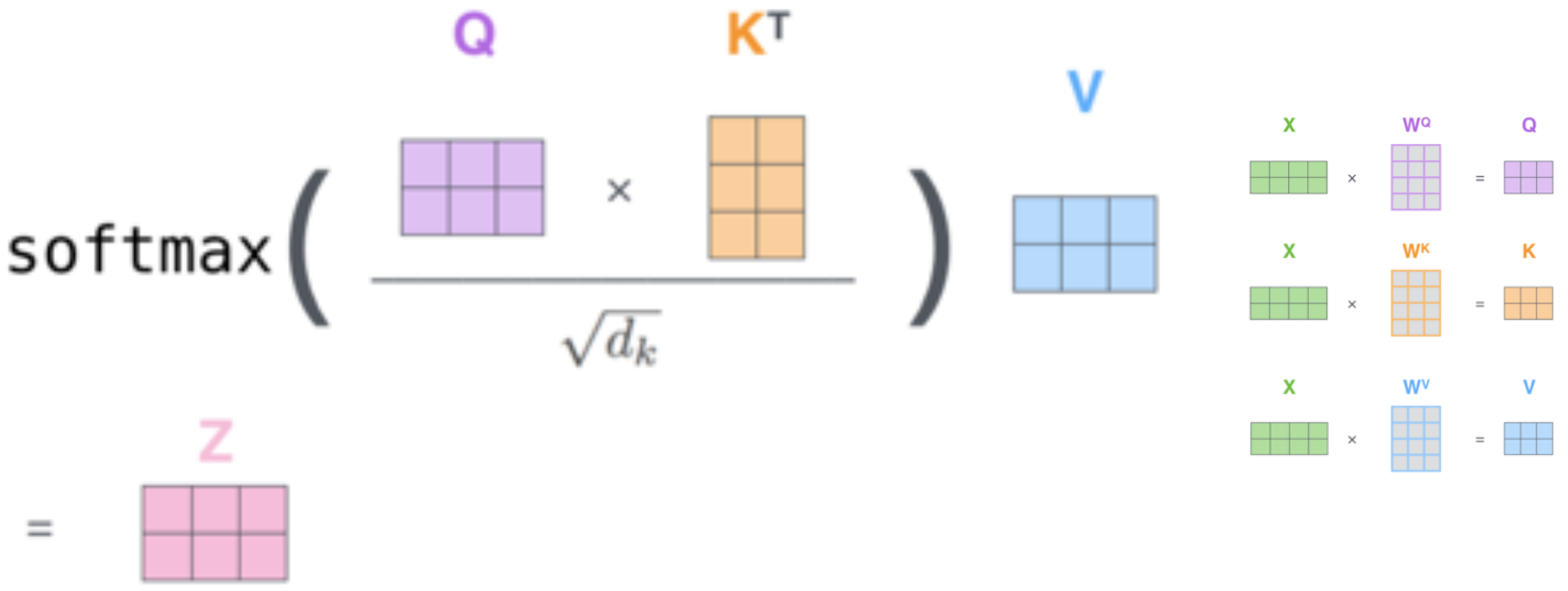
- Transformer
- Multi-head attention



- https://jalammar.github.io/

Self-Attention & Transformer

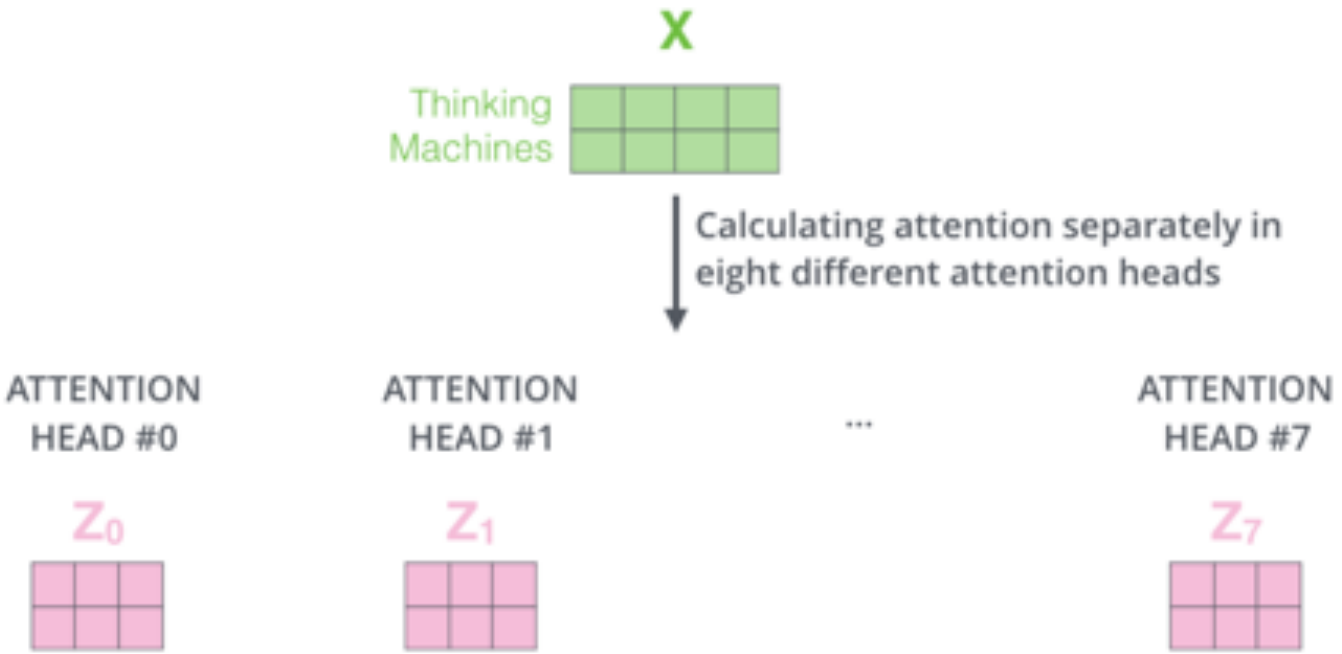
- Transformer
- Multi-head attention



- https://jalammar.github.io/

Self-Attention & Transformer

- Transformer
- Multi-head attention



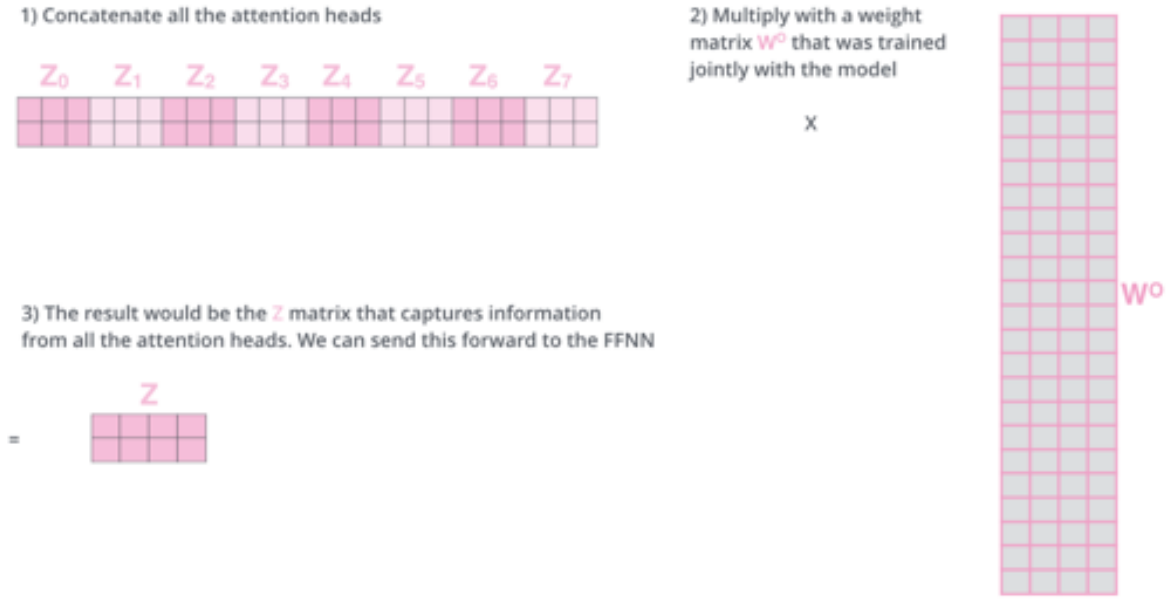
- https://jalammar.github.io/

Self-Attention & Transformer

- Transformer
- Multi-head attention

만들어진 head는 여러 개이지만 순방향 신경망을 통과할 수 있는 것은 오직 하나의 벡터

- 만들어진 여러 개의 [illegible]여 새로운 Z (head)를 제작



- https://jalammar.github.io/

1) This is our input sentence*

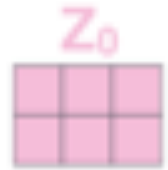
2) We embed each word*

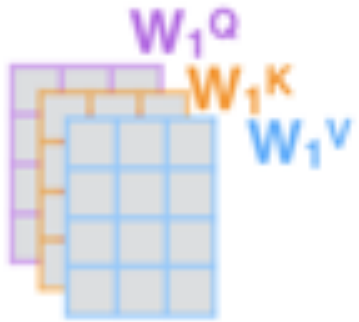
3) Split into 8 heads. We multiply X or R with weight matrices

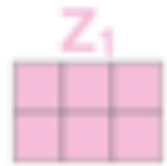
4) Calculate attention using the resulting Q/K/V matrices

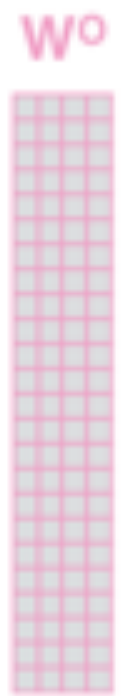
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

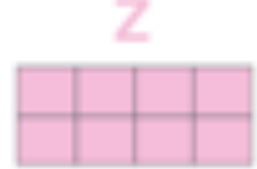
Thinking Machines



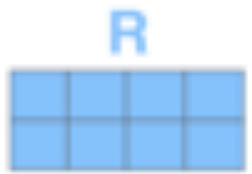


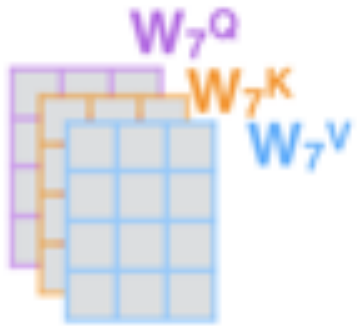


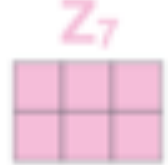




* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

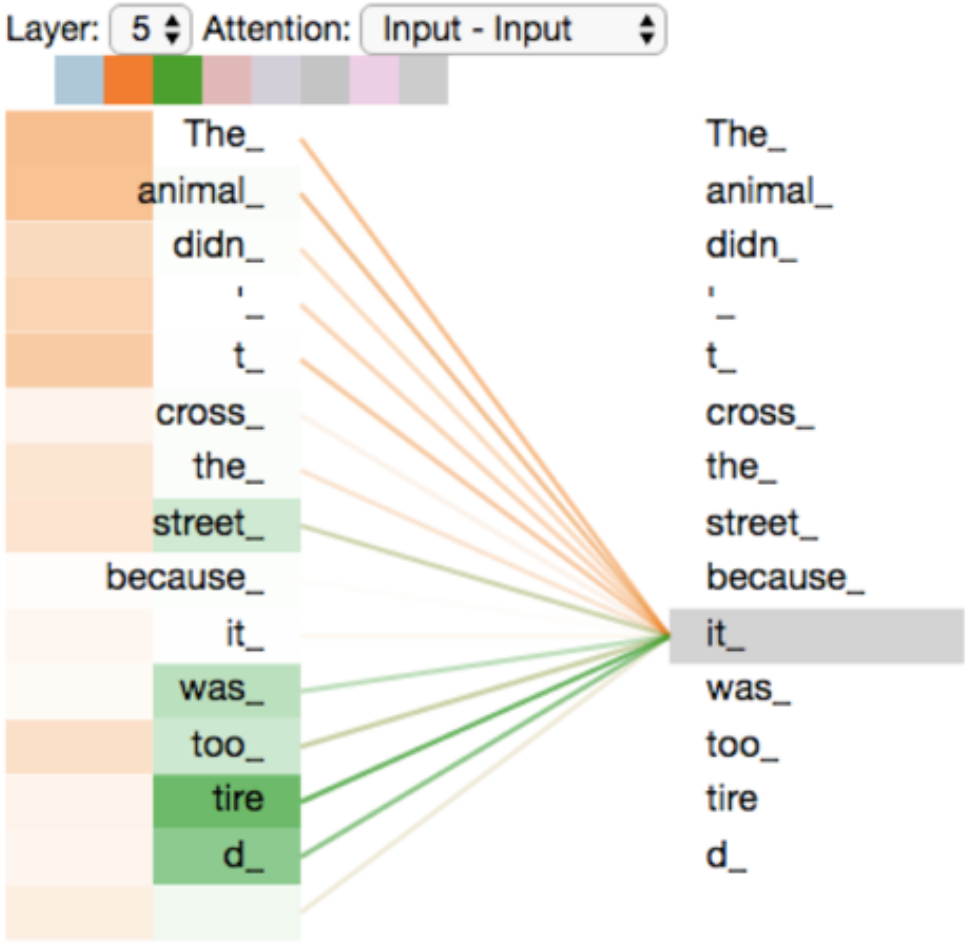






Self-Attention & Transformer

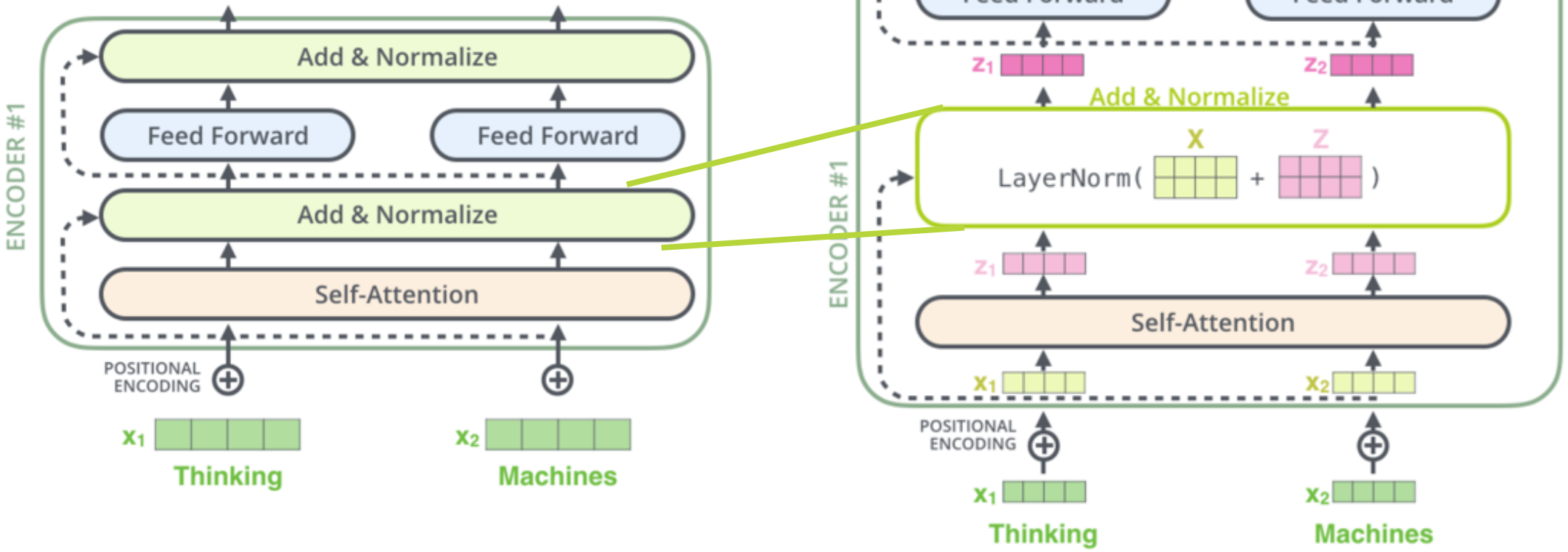
- Transformer
- Multi-head attention의 결과
- 주황색 attention은 it이 animal을 가르키지만, 초록색 attention은 it이 tire를 가르킴
- 다양한 representation이 가능



- https://jalammar.github.io/

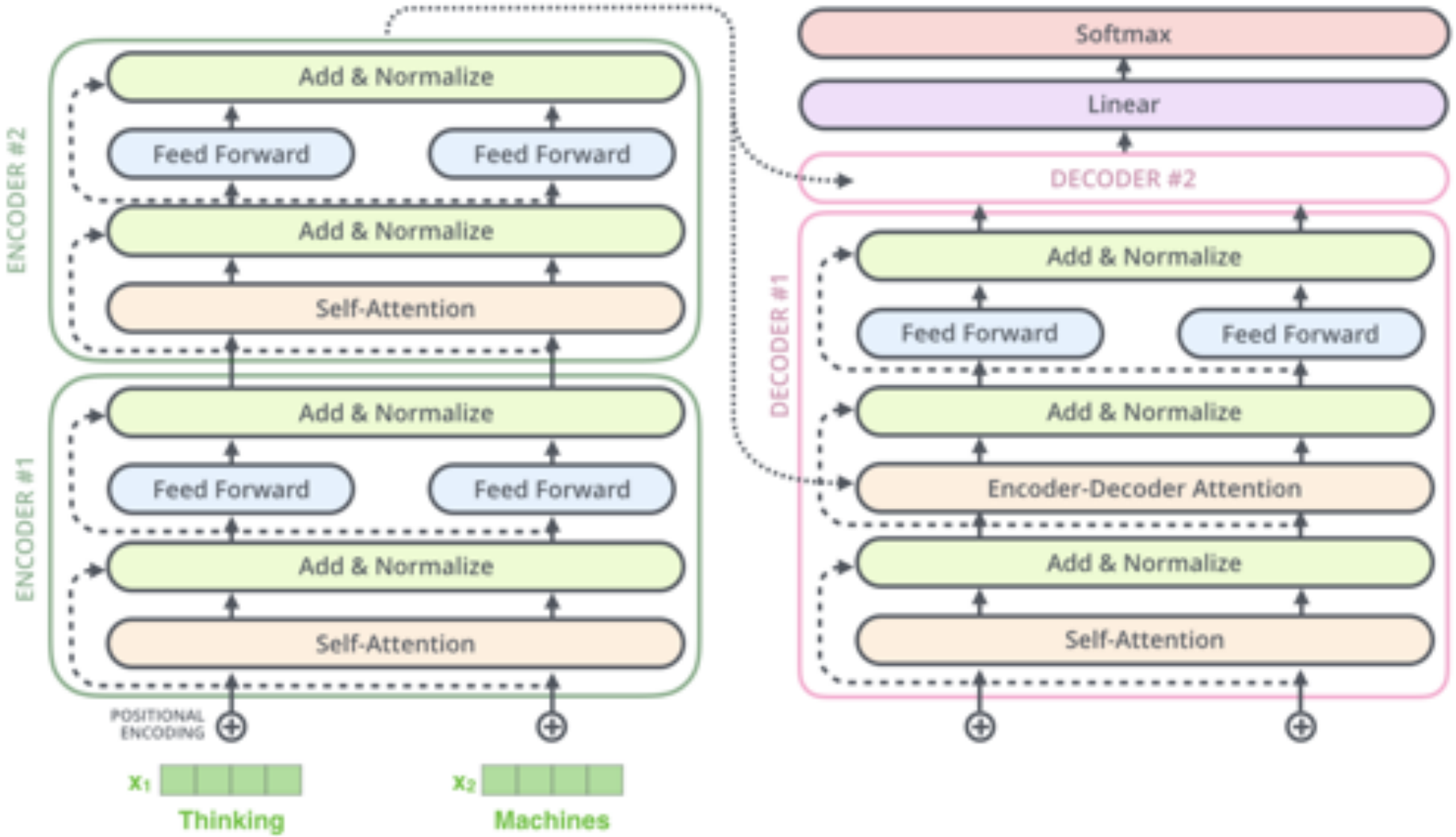
Self-Attention & Transformer

- Transformer



- https://jalammar.github.io/

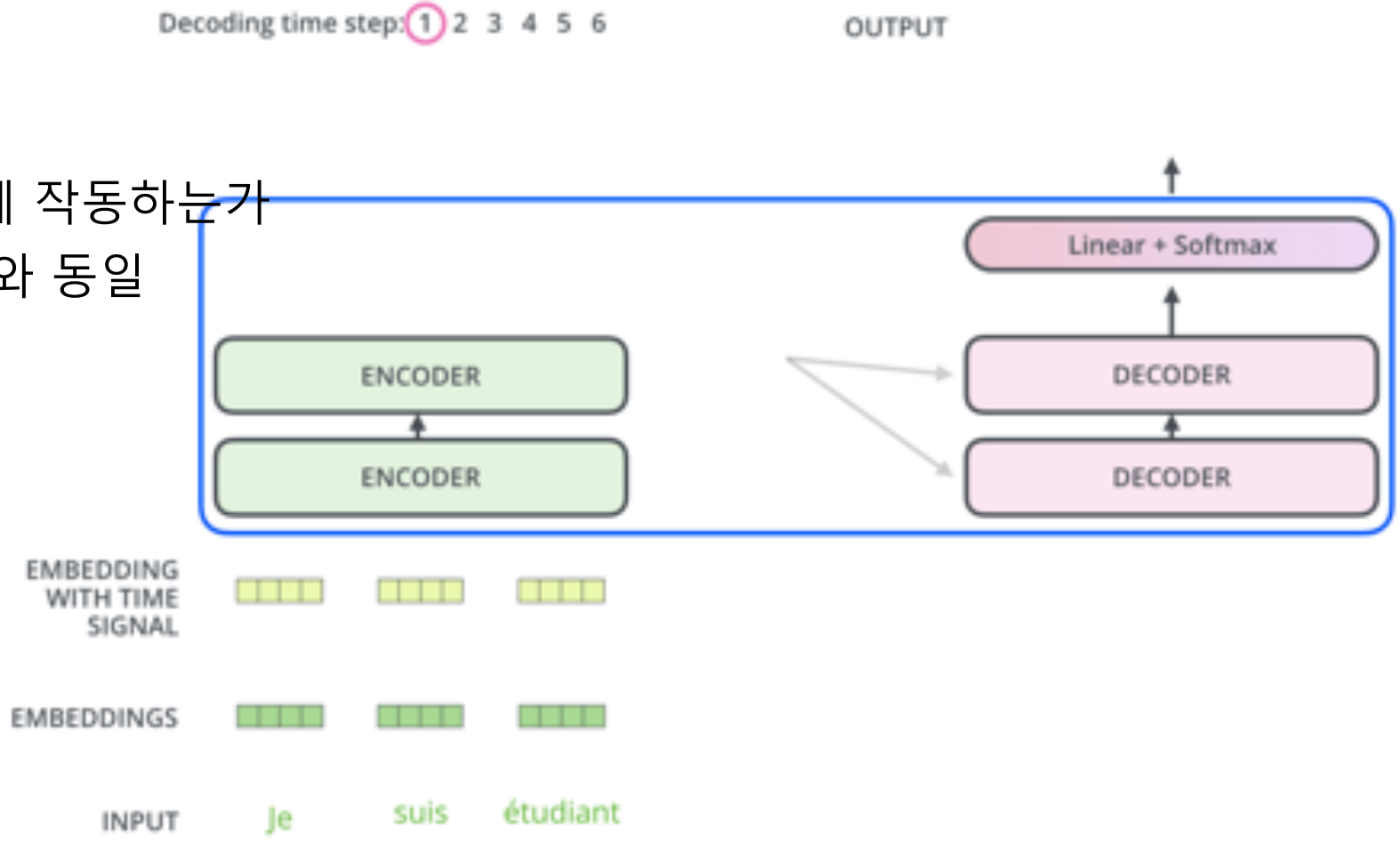
Self-Attention & Transformer



- https://jalammar.github.io/

Self-Attention & Transformer

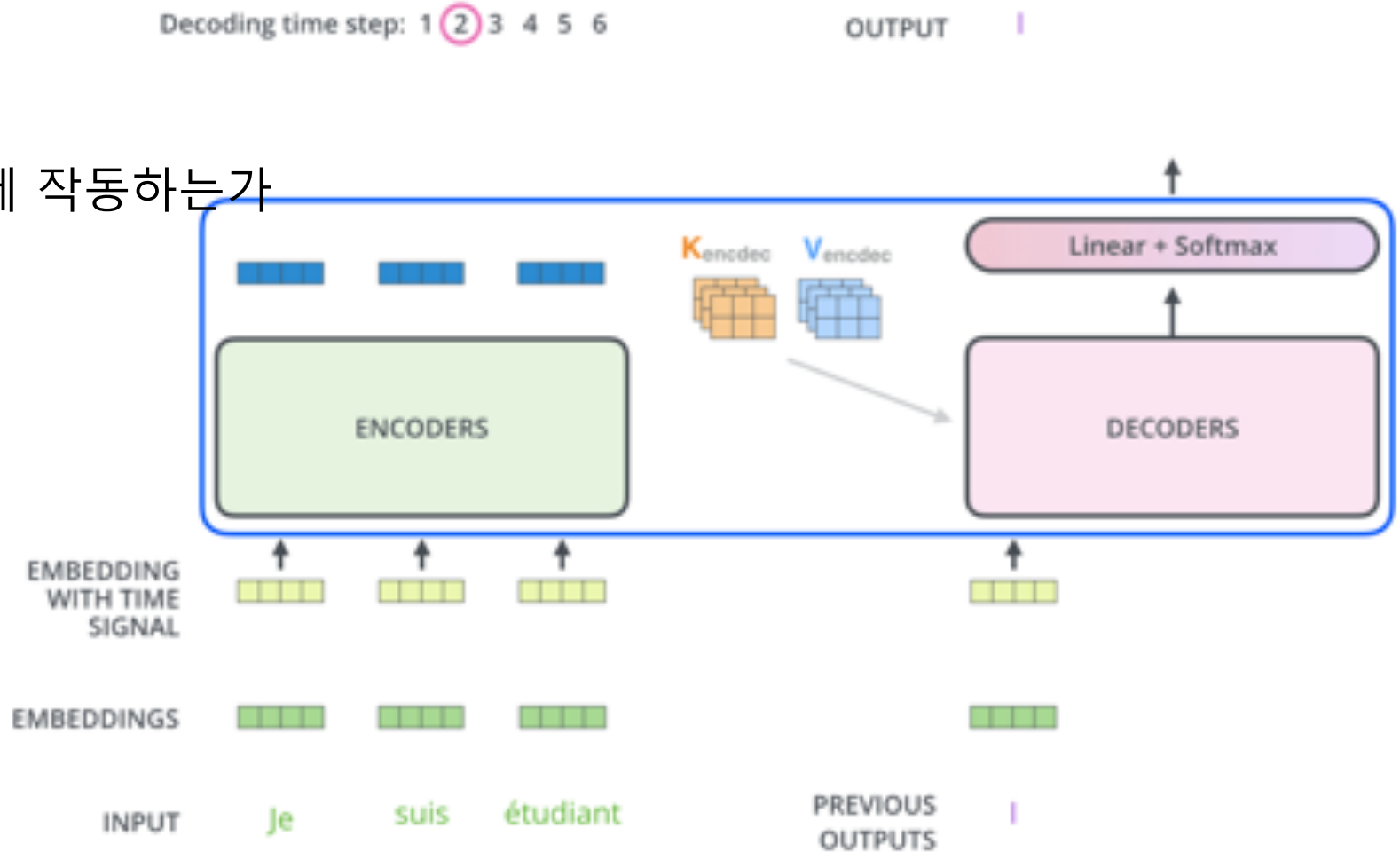
- Transformer
- 디코더가 어떻게 작동하는가
  :내부는 인코더와 동일



- https://jalammar.github.io/

Self-Attention & Transformer

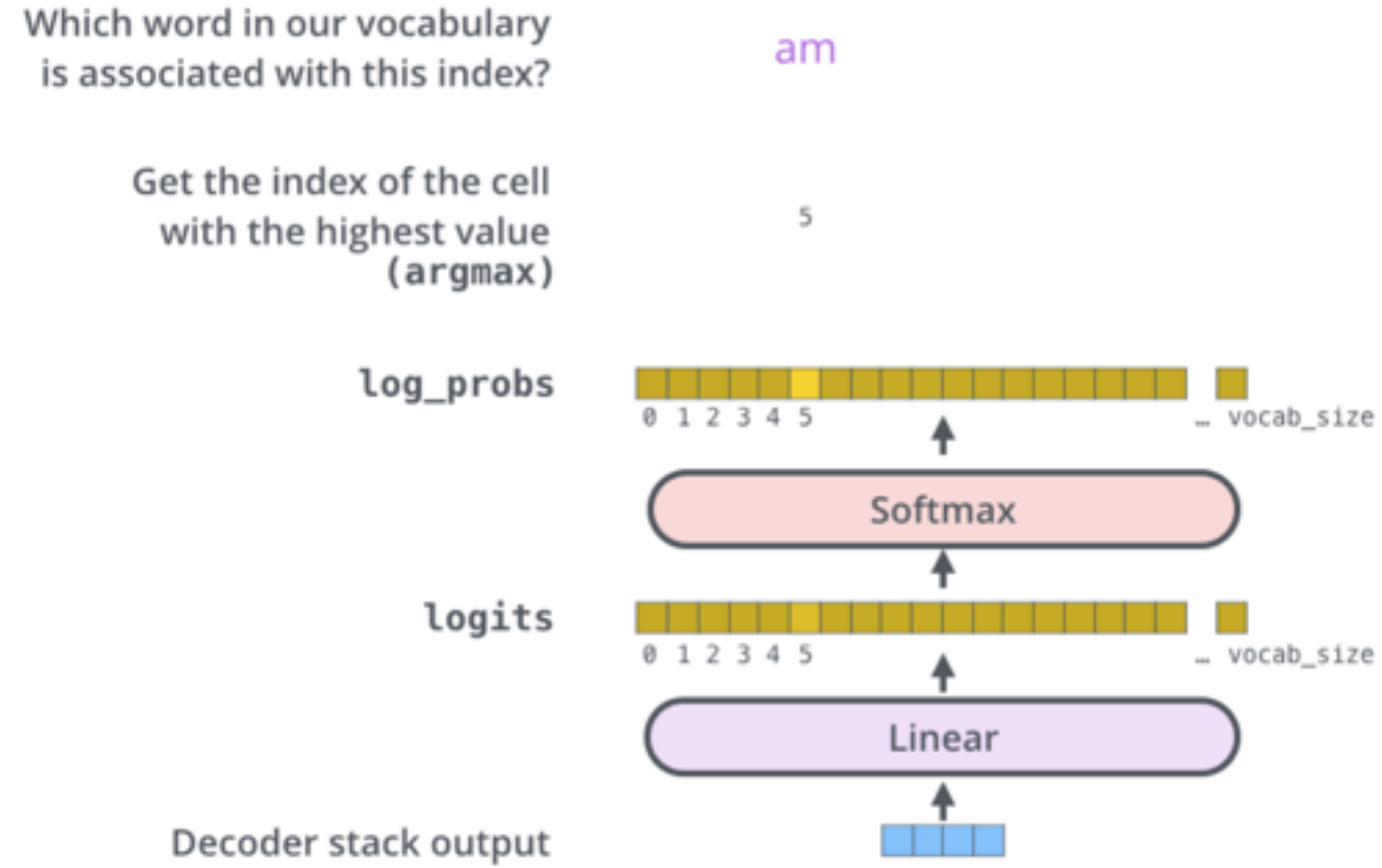
- Transformer
- 디코더가 어떻게 작동하는가



- https://jalammar.github.io/

Self-Attention & Transformer

- Transformer
- 디코더의 마지막
  Linear and Softmax Layer



- https://jalammar.github.io/

Self-Attention & Transformer

- 실제 Transformer의 학습: 기계 번역

Output Vocabulary

WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

Untrained Model Output

0.2	0.2	0.1	0.2	0.2	0.1

One-hot encoding of the word "am"

0.0	1.0	0.0	0.0	0.0	0.0

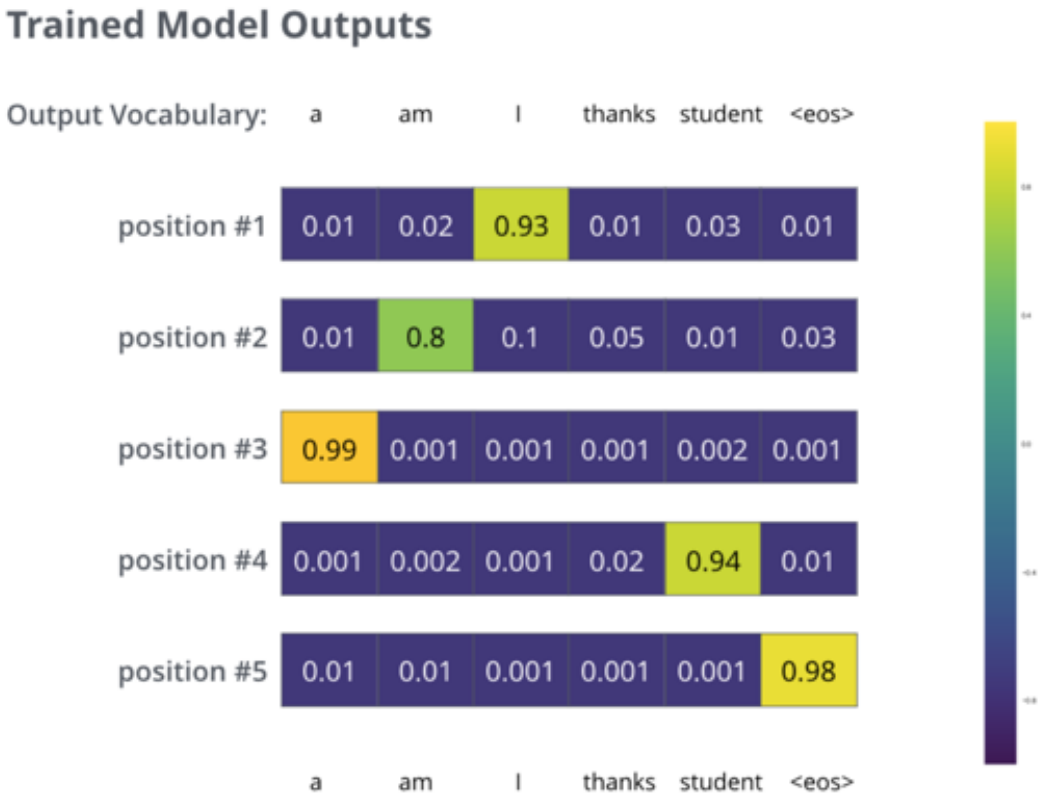
Correct and desired output

0.0	0.0	0.0	1.0	0.0	0.0
a	am	I	thanks	student	<eos>

- https://jalammar.github.io/

Self-Attention & Transformer

- 실제 Transformer의 학습: 기계 번역



- https://jalammar.github.io/

End