인공지능 : 기계가 사람의 행동을 모방하게 하는 기술

기계학습 : 데이터로부터 학습

딥러닝 : 인공신경망에 기반하여 많은 양의 데이터를 학습

빅데이터 : 데이터 관리, 수집, 시각화, 분석, 마이닝

기계학습 : 특징추출기-특징벡터-분류기(학습대상)

딥러닝 : feature extractor & classifier(학습대상) –파라미터가 있어 더 많은 데이터를 효과적으로 학습할 수 있다.

딥러닝 - 분류, 회귀, 물체 검출, 영상 분할, 영상 초해상도, GAN(예술적 창조물), 강화학습(ex.알파고 – 초기학습 후 경쟁을 해서 점점 강화되도록 구현) 가능

역전파 알고리즘 - 정방향으로 계산을 하고 정방향과 비교해서 에러를 내고 하나씩 역방향으로

전파

**얕은 신경망의 구조**

활성함수 – 비선형 특성

인공신경망 : 뉴런이 모여 서로 연결된 형태

fully-connected layer : 모든 뉴런이 서로 연결된 계층

얕은 신경망 : 입력, 은닉, 출력

얕은 신경망을 이용한 분류와 회귀

회귀 : 연속된 값의 출력을 추정하는 것

분류 : 특정 범주로 구분하는 작업

얕은 신경망의 동작은 출력 계층의 활성 함수에 의해 달라진다.

이진 분류-> 출력 한 개, 활성함수로 Sigmoid 함수 사용

다중 분류-> 출력 여러 개, Softmax(확률을 모두 더하면 1)

a는 활성함수 y = a(w^T \*x +b) 두 벡터의 내적

* y = a(Wx +b)

입력 계층 : 연산x, 다음 계층으로 넘기는 역할

계층의 크기 = 노드의 개수 = 입력 스칼라의 수 = 입력 벡터의 길이

은닉 계층 : 입력 계층과 연결된 전결합 계층

얕은 신경망에서 은닉 계층 1개만 사용

출력 계층 : 은닉 다음에 오는 전결합 계층

출력 계층의 크기 = 출력 스칼라의 수 = 출력 벡터의 길이

선형 회귀 : 데이터를 가장 잘 표현하는 선형식을 찾는 동작

다중 선형 회귀 -> y = w0x0+w1x1+b

입력관점으로 보면 회귀 곡선

**이진 분류 문제**

로지스틱 회귀 : 범주형 데이터를 대상으로 하는 회귀, 분류 기법

Sigmoid 함수 : 출력이 0~1사이, 2가지 클래스를 구분하기 위해 1개의 입력을 받는다.

교차엔트로피(CEE): 정확히 맞추면 오차는 0, 틀릴수록 오차가 무한 증가

얕은 신경망으로 classification 수행하면 출력 계층은 로지스틱 회귀와 동일

**다중 분류 문제 : 어떤 문제인지 표현**

원-핫 인코딩 : 한 개의 값만 1, 나머지는 0

희소 벡터 : 대부분의 값이 0, 크기가 있는 값이 희소하게 나타나는 벡터

softmax 함수 : 출력은 0~1사이 값

교차엔트로피(CEE) : 정답인 클래스에 대해서만 오차 계산

**모델의 학습과 최적화 이론**

손실 함수 : 알고리즘이 얼마나 잘못 하고 있는지 표현

regression(회귀)-평균 제곱 에러(MSE)

classification(분류)-교차 엔트로피 오차(CEE)

손실함수가 작을수록 잘 학습

학습에 필요한 요소 : data, model, loss

알고리즘 학습: 입력 값을 바꿔가면서 출력 값을 작아지게 함.

최적화 이론 : 출력 값을 가장 작게 하는 입력 값 찾기/ 전역 : 정의역 1개, 지역 : 정의역 여러 개 일 수 있다.

* 분석적 방법 : 함수의 모든 구간을 수식으로 알 때 사용하는 수식적 해석방법
* 수치적 방법 : 함수의 형태와 수식을 알지 못할 때 사용하는 계산적 해석방법

**경사 하강 학습법 : 경사를 따라 여러 번 스탭을 통해 최적점으로 다가간다.**

무차별 대입법(Brute-Force) : 가능한 모든 수를 대입

학습률 알파값에 비례해 이동, 적절한 학습률 중요

볼록함수 : 경사하강법으로 최적값에 도달 가능

비볼록함수 : 시작위치에 따라 다른 최적값을 가짐.

딥러닝 네트워크 학습 : 손실함수가 최소가 되게 하는 파라미터를 구하는 최적화 문제

saddle point : 기울기가 0이지만 극값이 아닌 지점

관성 : 이동 벡터를 이용해 이전 기울기에 영향을 받도록 하는 방법 / 이동벡터를 사용하므로 메모리 2배 사용

적응적 기울기(AdaGrad) : 변수별로 학습율이 달라지게 조절하는 알고리즘

기울기가 커서 학습이 많이 된 변수는 학습율을 감소시켜 다른 변수들이 잘 학습되도록 함. 학습

을 오래하면 안좋다.

RMSProp : AdaGrad의 문제점 개선

Adam : RMSProp + Momentum

**역전파 학습법**

심층 신경망(DNN): 얕은 신경망보다 은닉 계층이 많은 신경망

-은닉 계층 추가 = 특징의 비선형 변환 추가

-학습 매개변수의 수가 계층 크기의 제곱에 비례

-ReLU 도입

DP동적계획법 : 처음 계산할 때 값을 저장하므로 중복 계산X

순방향 추론 : 현재 매개변수에서의 손실값을 계산하기 위해 순차적인 연산을 수행

역전파 학습법(Back-Propagation) : 순방향 추론과 반대 / 연쇄법칙과 동적계획법을 이용해 효율적으로 계산 가능

Fully Connected Layer:행렬곱 연산으로 표현

수치 미분을 이용한 심층 신경망 학습

뉴런 구현 – 심층신경망 구현

합성곱 연산(CNN)과 이미지 필터

Dirac 델타 함수 : 모든 구간에서 적분한 값이 1

임펄스 응답 : LTI시스템에 임펄스를 입력 했을 때의 출력

합성곱 연산(Convolution) : 두 함수를 합성하는 합성곱 연산

LTI시스템은 입력 신호에 임펄스 응답을 합성곱한 결과와 같다.

이차원 신호: 0~1사이 실수로 된 2-D Signal로 표현

컬러 영상 : RGB의 3채널로 구성

영상의 합성곱 계산 – 필터를 한 칸씩 옮기면서 영상과 겹치는 부분을 모두 곱해 합친다.

잡음제거필터, 미분필터

뉴런 : 여러 개의 입력이 들어오면 가중치가 곱해져 summation 후 활성함수 이용

합성곱 : 입력 뉴런 대신 입력 영상, 가중치 대신 필터, 곱 대신 합성곱

합성곱 계층 : 합성곱으로 이루어진 뉴런을 전결합 형태로 연결, 특징이 나타나는 위치

합성곱 신경망 기본 구조 : 이미지 – 합성곱계층/풀링계층/활성함수

합성곱 계층에서는 영상의 크기는 그대로, 영상의 채널 수 달라짐

합성곱 계층에 의해 추출된 결과- 특징맵

풀링 계층 : 영상의 크기를 줄이기(Max Pooling, Average Pooling)

평탄화 : 입력된 특징맵의 화소를 나열해 하나의 벡터로 만드는 것 / 합성곱-전결합 연결 역할

Receptive Field : 풀링에 의해 작아진 특징 맵에 적용되면 원본 영상에서 차지하는 범위

일반 경사 하강법 – Gradient를 한번 업데이트 위해 모든 데이터 사용

확률적 경사 하강법 – Gradient를 한번 업데이트 위해 일부 데이터 사용

미니 배치 학습법 – 한번 에폭(전체를 한번 학습)마다 여러 배치(gradient를 구하는 단위)

Internal Covariate Shift : 계층별 입력 데이터 분포가 달라지는 현상

배치 정규화 : 배치별 평균과 분산을 이용해 정규화

학습 단계 : 배치에 맞게 정규화

추론 단계 : 이동평균 계산

심화 합성곱 신경망

Inception 모듈 : 다양한 크기의 필터를 적용(다양한 학습)해서 filter맵 출력

dimension reduction이 추가되면 연산량을 줄이기 위해 1\*1 합성곱 추가 -Bottleneck구조

Bottleneck구조 : Receptive field를 유지하면서 파라미터 수, 연산량 줄었다.

추가 분류기 사용 - 기울기 소실이 발생 방지

Residual Network :Skip Connection이 중요한 역할

Residual 구조 : 입력을 그대로 끌고와서 relu로 출력

Identity Mapping : relu가 사라지고 pre-activation제안

DenseNet : ResNet의 아이디어 계승

모델정의 Conv2D(출력채널개수,(커널크기), 패딩크기, activation))

#모델 정의만 수정하면 구현 가능

self.sequence.append(conv2d(16, (3, 3), padding='same', activation='relu'))

정의한 것을 call에서 연결, 학습 테스트 루프 정의는 이전과 같다.

def call(self, x, training=False, mask=None):

for layer in self.sequence:

x = layer(x)

return x

데이터셋 준비

# x\_train : (NUM\_SAMPLE, 28, 28) -> (NUM\_SAMPLE, 28, 28, 1) # 1채널인 영상인 것을 추가

x\_train = x\_train[..., tf.newaxis].astype(np.float32) #float64로 되어있어 float32로 바꿔준다.

x\_test = x\_test[..., tf.newaxis].astype(np.float32)

Residual Unit 구현

if filter\_in == filter\_out:

self.identity = lambda x: x #람다 함수를 이용해서 입력을 그대로 출력

else:

self.identity = tf.keras.layers.Conv2D(filter\_out, (1,1), padding='same')

def call(self, x, training=False, mask=None):

h = self.bn1(x, training=training) # BatchNormalization은 train할때와 test할 때 달라져야하므로

h = tf.nn.relu(h)

h = self.conv1(h)

Residual Layer 구현

class ResnetLayer(tf.keras.Model):

def \_\_init\_\_(self, filter\_in, filters, kernel\_size):

super(ResnetLayer, self).\_\_init\_\_()

self.sequence = list()

for f\_in, f\_out in zip([filter\_in] + list(filters), filters):

self.sequence.append(ResidualUnit(f\_in, f\_out, kernel\_size))

# [filter\_in]이 16이면 filters가 [32, 32, 32]

# zip([16, 32, 32, 32], [32, 32, 32])

def call(self, x, training=False, mask=None):

for unit in self.sequence:

x = unit(x, training=training)

return x

학습, 테스트 루프 정의

train\_step 함수에서 predictions = model(images, training=True)

test\_step함수에서 predictions = model(images, training=False)

순환신경망(RNN)

순차데이터 : 순서가 있고, 순서가 달라지면 의미가 손상

Resampling : Temporal Sequence ->Time Series로 변환

얕은신경망 – 대표적인 무기억 시스템/ 이전 입력에 영향을 받지 않는다.

기본적인 순환 신경망 – 얕은 신경망 구조에 순한이 추가 / 이전의 모든 입력에 영향

기본적인 순환 신경망은 기울기 소실문제가 있다.

LSTM(Long Short-Term Memory) : 기본적인 순환 신경망을 개선, 기억할 것은 오래 기억, 잊을건 빨리 잊는다. / Cell state(기억을 총괄하는 메모리 역할), hidden state, forget gate, input gate, output gate

GRU-LSTM을 간소화한 버전 , cell state가 없고, hidden state만 존재

reset gate(이전 hidden state를 얼마나 사용할지 정하는 역할) 추가

시간펼침 역전파 (Back Propagation Through Time)학습법

시간적으로 펼쳐 둔 상태에서 역전파를 한다. 시간적으로 펼쳐진 변수들은 동일한 변수다.

다중입력 단일출력 / 단일입력 다중출력/ 다중입력 다중출력(문장 번역, 챗봇)

RNN의 입력 텐서: 길이가 L보다 짧은 경우, 앞을 0으로 채운다.

RNN의 출력 텐서: 길이가 L보다 짧은 경우, 뒤에 0으로 채운다.

순환신경망의 학습법

BPTT : N\*L\*I(학습 데이터 입력) , 다중입력 다중출력은 Truncated BPTT(순차 데이터의 길이를

일정한 T길이로 잘라서 사용- 메모리 사용이 줄어드는 장점이 있다.) 길이 T로 쪼개진 Truncation사이에서 기울기 역전파가 이루어지지 않는다.

Attention기법의 이해

Sequence-to-sequence model : encoder->(context전달)->sos->eos->decoder

영어 문장의 데이터화 : Toekenizer(영어->숫자), embedding(특정한 범위에 원핫인코딩해서 넣고 실수화)

한글 문장의 데이터화 : 형태소 분석, 문장부호 제거, Toekenizer, embedding

attention기법 : decoder단에서 어떤 encoder 정보에 집중하는지 알고 있는 기법

Attention신경망

query : 질의 , 찾고자 하는 대상

key : 키, 저장된 데이터를 찾고자 할 때 참조하는 값

value : 값, 저장되는 데이터

dictionary : key-value pair로 이루어진 집합

attention mechanism : query에 대해 어떤 key가 유사한지 비교하고 유사도를 반영해 value와 합성

대부분 attention network에서 key와 value를 같은 값 사용

seq2seq에서 인코더의 hidden layer들을 key와 value로 사용

seq2seq-query : 디코더의 hidden layer , 하나 앞선 time-step의 hidden layer 사용

seq2seq-application : RNN으로 hidden layer를 입력하기 전, attention value를 concatenate하여 입

력

**Attention is really all you need**

**The Transformer** : 번역 문제에 RNN, CNN 사용x, attention만 사용해 state-of-the-art 성능을 끌어낸 연구/ scaled dot-product attention과 multi-head attention 알고리즘의 핵심

네트워크 특성 : seq2seq와 유사한 transformer 구조 사용, 병렬 계산 가능, 입력된 단어의 위치를 표현해줘야함.

positional Encoding : 시간적 위치별로 고유의 코드를 생성해 더하는 방식

scaled dot-product attention : query, key-value의 구조를 띄고 있다.

multi-head attention : linear 연산을 이용해 q, k, v의 차원을 감소

position-wise feed-forward : fully-connected layer – ReLU – fully-connected layer로 구성

output softmax : linear 연산을 이용해 출력 단어 종류의 수에 맞춤

import Okt : 형태소 분석

okt.morphs(line)) : 형태소 분석

attention일 때 decoder에 추가

S\_ = tf.concat([s0[:, tf.newaxis, :], S[:, :-1, :]], axis=1)

A = self.att([S\_, H])

y = tf.concat([S, A], axis=-1)

attention일 때 그 전보다 에러가 줄어든다.

과적합의 해결

데이터 셋의 의미 : 학습데이터 – 실제 모델을 학습, 검증 데이터(과대 적합을 막기 위해 사용) – 학습에 사용하지 않고 학습이

잘 되는지 검증 시 사용, 테스트 데이터 – 학습을 마친 모델을 평가하기 위해 한번 사용

과대적합(overfitting) 막기 위한 방법

1. Early Stopping : validation loss가 여러 epoch동안 감소하지 않으면 overfitting으로 간주해

학습 중단

1. drop out : 지정한 비율의 뉴런을 제거하고 학습하는 방법, 테스트 시 모든 뉴런을 사용해서 여러 network를 ensemble한 효과O
2. 배치 정규화 : 중간 Feature들을 그대로 사용하지 않고 변경하여 학습해서 overfitting 개선

* 전결합 계층(각 뉴런 별로 정규화)과 결합 -> 실제 테스트 시, Batch Norm에 영향X
* 합성곱 계층에 적용 -> 채널별로 정규화, 배치 높이 너비에 대해 평균과 분산 계산
* 배치 정규화의 한계 : mini-batch에 의해 크게 영향을 받는다./ 너무 작거나 너무 크면

잘 작동X-> 개선 : 1)가중치 정규화 – 학습 시 CNN에서 배치 정규화 대비 연산량이 매우 감소, 2) 계층 정규화 : 계층에 대해 정규화, 배치의 크기에 영향x, RNN에서도 잘 동작

추가적인 손실 함수

정규화 기법 : 알고리즘의 일반화를 개선하려는 모든 기법, loss값은 오히려 증가, validation과 test에서 loss값이 감소하지 않도록 하는 기법 / 최적화에 추가적인 손실 함수를 추가하는 것

weight decay : weight의 l-2 Norm을 최소화하는 정규화 기법 / weight가 지나치게 커지는 것을

막는다.

영상복원문제 -> 결과물에 대해 주관적인 평가가 중요하기 때문에 정규화를 적용하기 좋다.

VGG Loss : VGG-16 Network의 특징을 사용하여 구현된 Content Loss를 많이 사용

SSIM (화질 평가 기법) : SSIM이 최대가 되도록 하는 Loss를 추가하여 MSE에서 보지 않는 구조적

인 특성 반영

데이터 증강 기법

불변성 : X-불변 함수는 입력에 X를 적용해도 출력에 X를 적용한 것과 같다.

Affine Transform : 영상의 2차원 변환 (Rotation, Shearing(밀림), Translation(이동), Scaling(크기)

CNN은 affine transform에 취약하다.(affine transform으로 변환된 영상은 다른 영상으로 인식)

데이터 증강 시, affine transform을 적용해서 학습을 하는게 좋다.

자가 정규화 신경망(Self-Normalizing Neural Network) : 바닐라 CNN에 약간의 변화를 주면서, 스스로 정규화하는 계층 형성

SELU(Scaled Exponential Linear Unit) : ReLU와 달리 음수 값을 Exponential 하게 활성화하는 특징o

->DropOut : ReLU 에 잘 동작하는 것을 반영, SELU에 적합하도록 변형

SMOTE 알고리즘 : 데이터가 불균형할 때 어떻게 할것인지

임의 언더 샘플링 : 다수 클래스에서 임의로 샘플링하여 크기를 맞추는 방법

임의 오버 샘플링 : 소수 클래스의 데이터를 반복하여 양을 학습 데이터의 양을 맞추는 방법

SMOTE : k개의 가장 가까운 layer 중 랜덤으로 하나의 샘플을 선택하여 linear combination을 추가

임의 오버 샘플링 보다 효과적 / 이를 보완한 것이 borderline-STMOTE : 위험 지역인 경계에 있

는 샘플만 오버 샘플링

L-2정규화(Ridge): 가중치의 L-2 Norm을 최소화하는 방법

L-1정규화(Lasso) : 가중치의 절대값에 패널티를 줌 , 가중치의 L-1 Norm을 최소화 하는 방법

Lasso – 가중치값을 정확하게 0으로 만든다, 중요한 특징을 선택하는 효과

Ridge – 큰 가중치의 값을 작게 만든다.

MSE Loss : 데이터의 평균을 나타냄, 데이터를 Smoothing 하는 효과

MAE Loss : 에러가 커져도 동일한 패널티, 데이터의 중간값을 나타냄

**딥러닝 트렌드 (PART 5)**

classification : 주어진 데이터를 주어진 클래스(라벨)에 의해 분류하는 법 학습/ discrete한 output

을 가짐

clustering : 주어진 데이터를 데이터의 특징에 의해 스스로 클래스로 분류/ 연속적인 output을 가짐

class 의 수 : 나누고자 하는 종류의 수

전처리 : data에서 유의미한 특징들을 뽑아 input으로 넣기

End-to-End : 전처리 과정, feature engineering없이 데이터만 넣으면 알아서 학습해 input을 넣으면 output이 나옴

Logistic Regression – 이진 데이터

Non Linear 를 Kernel Trick을 이용해서 Linear로

SVM- supervised learning

K Means Clustering – unsupervised learning

Drop out – overfitting 방지

이미지 처리 분야 딥러닝 모델

인풋 변화에 민감한 MLP는 일반적인 학습이 어렵다.

변하지 않는 특징 학습->CNN

CNN : Feature extraction – receptive field, convolution filter / shift and distortion invariance – subsampling(max pooling)/ classification(fully connected output)

AlexNet : conv layer, max-pooling layer, dropout layer 5개, fully connected layer 3개 , nonlinearity function : ReLU, batch stochastic gradient descent, 1000 output classification /

1. 2개의 병렬구조가 특징

2. ReLU 함수 사용 : 같은 정확도 유지하면서 Tanh 사용했을때 보다 6배 빠르다.

3. Dropout 사용 : Overfitting 방지 위해 사용, fully-connected layer의 뉴런 중 일부를 생략하면서 학습 진행, dropout은 훈련시 적용, 테스트 시에 모든 뉴런 사용

4. Overlapping Pooling LeNet-5의 경우 non-overlapping average pooling이 사용된 반면, AlexNet에서는 overlapping max pooling 사용

5. Data Augmentation : 과적합을 막기 위해

6. local response normalization(LRN)

ZFNet : Rotation보다 Translation, Scaling 에 상대적으로 강인, 앞쪽보다 뒤쪽 layer에서 변화에 invariant, 이미지의 일부를 가려가면서 여러 번 feature map의 activation 정도를 통해 heatmap 그림

Network in Network : 네트워크 안에 네트워크 ,내부에 있는 MLP를 1\*1 convolution을 통해 채널간 연산을 하게 함.

자연어 처리: 일상 생활에서 쓰이는 언어

NLP(Natural Language Processing) : 자연어의 의미를 분석하여 컴퓨터가 처리하도록 하는 일

ex) 챗봇, 클린봇, 개인비서서비스

NLP가 아닌 것 : 자동 음성 인식, 광학 문자 인식

문장 번역 – 문장 번역 기능을 학습할 수 있는 Seq2seqmodel

감정분석 – 금융 또는 경영에서 판단에 도움을 받기 위해 사용

챗봇 – 문맥 이해, 다양한 답변 생성해야함.

문맥광고 – 컨텐츠를 분석하여 광고와 매칭

**단어를 숫자로 표현하기**

밀집 표현 : 희소 표현된 단어를 임의의 길이의 실수 벡터로 표현할 경우

희소-> 밀집 : word embedding /밀집 표현된 결과 : 임베딩 벡터

말뭉치(corpus) : 분석의 용이성을 위해 형태소 분석 포함.

Word2Vec : 가장 많이 사용되는 Word embedding 방법 / 유사한 의미를 가진 단어는 유사한 벡터가 되는 특징

CBOW (Continuous Bag-of-Words) : 주변의 단어로 현재 단어를 추정하는 방법

CBOW의 수식 : Projection layer, output layer로 구분

Projection Layer : one-hot vector의 특성상, LUT(Look-Up Table)의 형태로 구현

Skip-Gram : Window 중앙에서 주변 단어를 추정하는 방식

형태소 분석기 : 문장을 형태소 단위로 구분하고, 언어적인 구조를 파악하는 것

KoNLPy – 한글 형태소 분석기를 통일된 방법으로 쉽게 사용 가능

Seq2seq(Attention) : 인코더와 디코더로 연결된 모델이 context로 연결, 번역문제를 학습하기 위해 널리 사용되고 있는 RNN구조

context를 개선방법 : encoder hidden state를 모아 decoder로 전달

Dictionary : key-value pair로 이루어진 집합(“2019”:”EndGame”)

attention value : query와 key를 비교해서 value를 합성

seq2seq에서 encoder의 hidden layer들을 key와 value로 사용, decoder의 hidden layer들을 query로 사용

ConvNet(합성곱 신경망)을 이용한 분류

합성곱 계층 : 합성곱으로 이루어진 뉴런을 전결합 형태로 연결한 것

Transormer : seq2seq와 유사한 transformer구조, RNN의 BPTT와 같은 과정이 없으므로 병렬 계산 가능, 제안하는 scaled dot-product attention과 이를 병렬로 나열한 multi-head attention블록이 알고리즘 핵심, 입력된 단어의 위치를 표현하기 위해 positional encoding 사용

-positional encoding : 시간적 위치별로 고유의 code를 생성하여 더하는 방식, 전체 sequence의

길이 중 상대적 위치에 따라 고유의 벡터를 생성하여 emedding된 벡터에 더해줌.

-scaled dot-product attention : query, key-value 구조를 띄고 있다.

-multi-head attention : linear 연산을 이용해 Q, K, V 의 차원을 감소, h개의 attention layer를 병렬

적으로 사용

-position-wise feed-forward : fully-connected layer, ReLU, fully-connected layer로 이루어짐.

-Add & Norm : Forward-Path와 Skip-Connection을 더한 후 layer Normalization 해주기

-output softmax : linear연산을 이용해 출력 단어 종류의 수에 맞춤.

실습

Dot-Scaled Attention

self.q = Dense(d\_reduced, input\_shape=(-1, d+emb)) #(배치에 따라 다르다, 임베딩)

self.k = Dense(d\_reduced, input\_shape=(-1, d+emb)) #키

self.v = Dense(d\_reduced, input\_shape=(-1, d+emb)) #value

self.scale = Lambda(lambda x : x/np.sqrt(d\_reduced))

self.masked = masked

q = self.scale(self.q(x[0])

k = self.k(x[1])

v = self.v(x[2])

k\_T = tf.transpose(k, perm = [0, 2, 1]) #배치는 그대로 유지 나머지 transpose

comp = tf.matmul(q, k\_T)

comp += mask

comp = tf.nn.softmax(comp, axis=-1)

return tf.matmul(comp, v)

Multi-Head Attention

super().\_\_init\_\_()

self.attention\_list = list()

for \_ in range(num\_head):

self.attention\_list.append(DotScaledAttention(d\_emb, d\_reduced, masked)

self.linear = Dense(d\_emb, input\_shape=(-1, num\_head\*d\_reduced)

def call(self, x, training=None, mask=None):

attention\_list = [a(x) for a in self.attention\_list]

concat = tf.concat(attention\_list, axis =-1)

return self.linear(concat)

Encoder

def\_\_init\_\_(self, num\_head, d\_reducd):

super().\_\_init\_\_()

self.num\_head = num\_head

self.d\_r = d\_reduced

def.build(self, input\_shape):

self.multi\_attention =MultiHeadAttention(self.num\_head, input\_shape[-1], self.d\_r)

self.layer\_norm1 = LayerNormalization(input\_shape=input\_shape)

self.dense1= Dense(input\_shape[-1]\*4, input\_shape=input\_shape, activation=’relu’)

self.dense2=Dense(input\_shape[-1], input\_shape=self.dense1.compute\_output\_shape(input\_shape))

self.layer\_norm2 = LayerNormalization(input\_shape=input\_shape)

super().build(input\_shape)

def call(self, x, training=None, mask=None):

h = self.multi\_attention(x, x, x))

ln1 = self.layer\_norm1(x+h)

h = self.dense2(self.dense1(ln1))

return self.layer\_normal(h+ln1)

def compute\_output\_shape(self, input\_shape):

return input\_shape

Decoder

def\_\_init\_\_(self, num\_head, d\_reducd):

super().\_\_init\_\_()

self.num\_head = num\_head

self.d\_r = d\_reduced

def.build(self, input\_shape):

self.self\_attention =MultiHeadAttention(self.num\_head, input\_shape[0][-1], self.d\_r)

self.layer\_norm1 = LayerNormalization(input\_shape=input\_shape)

self.multi\_attention =MultiHeadAttention(self.num\_head, input\_shape[0][-1], self.d\_r)

self.layer\_norm2 = LayerNormalization(input\_shape=input\_shape)

self.dense1= Dense(input\_shape[0][-1]\*4, input\_shape=input\_shape[0], activation=’relu’)

self.dense2=Dense(input\_shape[0][-1], input\_shape=self.dense1.compute\_output\_shape(input\_shape[0]))

self.layer\_norm3 = LayerNormalization(input\_shape=input\_shape)

super().build(input\_shape)

def call(self, inputs, training=None, mask=None):

x, context = inputs

h = self.self\_attention(x, x, x))

ln1 = self.layer\_normal(x+h)

h = self.multi\_attention(ln1, context, context))

ln2 = self.layer\_morm2(ln1+h)

h = self.dense2(self.dense1(ln2))

return self.layer\_norm3(h+ ln2)

def compute\_output\_shape(self, input\_shape):

return input\_shape

def compute\_output\_shape(self, input\_shape):

return input\_shape

Transformer Architecture

class Transformer(Model):

def \_\_init\_\_(self, src\_vocab, dst\_vocab, max\_len, d\_emb, d\_reduced, n\_enc\_layer, n\_dec\_layer, num\_head):

super(). \_\_init\_\_()

self.enc\_emb=Embedding(src\_vocab, d\_emb)

self.doc\_emb =Embedding(dst\_vocab, d\_emb)

self.pos\_enc=PositionalEncoding(max\_len, d\_emb)

self.encoder = [Encoder(num\_head, d\_reduced) for \_ in range(n\_enc\_layer)

self.decoder = [Decoder(num\_head, d\_reduced) for \_ in range(n\_dec\_layer)

self.dense = Dense(dst\_vocab, input\_shape=(-1, d\_emb))

def call(self, inputs, training=None, mask=None): #(src\_sentence, dst\_sentence\_shift)

src\_sent, dst\_sent = inputs

h\_enc = self.pos\_enc(self.enc\_emb(src\_sent))

for enc in self.encoder:

h\_enc = enc(h\_enc)

h\_dec = self.pos\_enc(self.doc\_emb(dst\_sent\_shifted))

for dec in self.decoder:

h\_dec = dec([h\_dec, h\_enc])

return tf.nn.softmax(self.dense(h\_doc), axis=-1)

자연어처리 – 정량 지표

손실 함수 : 최적화 이론에서 최소화 하고자 하는 함수, 알고리즘 학습 중 아직 ‘얼마나 못하는지’표현, 보통 미분 가능한 함수를 사용, 학습 과정에서 중요한 역할을 함.

성능 척도 : 학습된 알고리즘의 성능을 측정하는 지표, 정량적으로 알고리즘을 비교/평가하기 위함, 미분 가능 여부에 관계 없이 사용, 학습 과정에서 사용x, ‘비즈니스 목표’와 연관 높음

Cosine Similarity : 두 가지 벡터를 비교할 때 유용

Modified Unigram Precision = (Ca에 나온 단어 중 Ref에 있는 단어의 수 – Ref를 넘어서는 횟수)/ Ca에 있는 단어의 수

BLEU:N개의 n-Gram Precision을 합산하는 형태