2020/7/10

-이해한 내용

다변수 상황에서 점곱을 이용해 식을 간편하게 쓰는 방법을 배웠다.

hypothesis를 행렬을 이용하여 간단하게 표현하는 법을 배웠다.

logistic 데이터와 linear 데이터의 차이를 이해했다.

dicision boundary 를 통해 어떤 선형적인 값을 로지스틱하게 바꾸는 방법을 알았다.

-이해하지 못한 내용

8번째 영상을 보면 1900번째에서 2000번째로 갈때 오히려 cost가 증가하였는데

왜 그런 것인지 잘 이해가 되지 않았다.

2020//7/12

-이해한점

시그모이드 함수에 대하여 이해했다.

binary classification 의 과정에 대하여 배웠다.

binary classfication을 여러번 반복하여 multinmial classfication을 하는 방법을 이해했다.

-이해하지 못한 점

어차피 영역으로 나누는 거면 multinomial classfication을 굳이 binary classfication을 여러번 반복하는 형식으로만 구현해야되는 것인지 약간의 의문이 들었다.

2020/7/13

-이해한 부분

cross-entropy cost function 에 대하여 배웠다.

score을 구해 함수를 통해 확률을 구하는 과정을 배웠다.

-이해하지 못한 부분

tf.tensor() 가 정확히 어떤 역할을 하는 함수인지 잘 모르겠다.

one\_hot의 동작이 잘 이해가 가지 않았다. (검색해서 찾아봐야겠다)

2020/7/14

-이해한 점

learning rate decay 기법에 대하여 이해했다.

표준화 기법과 정규화 기법에 대하여 알았다.

overfit과 underfit 문제가 무엇인지 이해했다.

feature 을 줄이고 늘리고, 차원을 줄이는 방법 등을 통해 데이터 전처리를 한다는 것을 배웠다.

-이해하지 못한 점

noisy data의 기준이 무엇인지 잘 모르겠다.

프로그램이 스스로 noisy data가 무엇인지 어떻게 알고 구분하는 것인지 궁금하다.

2020/7/15

학습과 평가를 위한 데이터를 잘 구성해야되는 이유에 대하여 이해했다.

online 과 batch 학습의 차이를 알았다.

fine tuning과 feature etraction 이 무엇인지에 대하여 배웠다.

뉴론에서부터 시작된 인공지능 연구의 역사에 대하여 알게되었다.

convolutional neural network에 대하여 배웠다.

자연어 데이터의 전처리 과정이 잘 이해되지 않는다.

2020/7/16

cifar 라는 단체가 어떤 연구를 했는지 어떤 단체인지 알게되었다.

딥러닝이라는 단어의 등장 기원을 알게 되엇다.

시그모이드 함수를 이용하고 다중으로 설계하여 xor을 해결하는 과정을 배웠다.

backpropagation이 무엇인지 배웠다.

chain rule에 대하여 다시 복습했다.

2020/7/17

logistic regression을 여러층으로 하여 만든 neural network가 무엇인지 배웠다.

tensorboard 가 무엇인지 배웠다.

sigmoid의 문제가 무엇인지와 Relu함수가 무엇인지와 사용 이유를 알게되었다.

sigmoid 를 이용할 때, 값이 boundary와 일치하면 어떤 결과가 나오는지 의문이 생겼다.

spyder을 이용해 실습을 진행해 보았다.

2020/7/18

weight 초기화가 무엇인지 왜 해야하는 지 알았다.

xavier initialization과 relu 에서 쓰이는 he initialization 에 대하여 배웠다.

over-fitting이면 왜 Test 샘플들에 대한 예측이 정확하지 않은지 이해했다.

dropout이 효과적인 이유가 잘 이해가 가지 않는다.

dropout을 할때 한 layer의 노드가 전부 사용하지 않음으로 될 경우 어떻게 되는 것인지 궁금하다.

nomalization의 원리가 좀 햇갈렸다.

2020/7/19

convNet이란 무엇인지 배웠다

padding이 무슨 의미인지 알게되었다.

서로다른 weight로 convolution layer을 형성하는 원리를 알았다.

pooling이 무엇인지 그중에서도 max pooling이 무엇인지 배웠다.

2020/7/22

2D Convolution의 과정에 대하여 배웠다.

zero padding에 대하여 배웠다.

tf.keras.layers.Conv2D api에 대하여 배웠다.

2020/7/23

모두를 위한 딥러닝 2의 40, 41,42,43을 학습하였습니다.

헷갈리는 용어 와 잘 이해가 안가는 용어들은 구글에 검색하여 나무위키와 같은 사전을 참고하였습니다.

2020년 7월 23일 6시 부터 8시까지 2시간 동안 학습하였습니다.

이전 결과값을 가져와 활용하는 RNN에 대하여 학습하였습니다. 플립플롭 비슷하다고 생각하니 무엇인지 이해하기는 쉬웠습니다.

many to one 에 대하여 학습

40강에서 CNN에 대하여 듣는데 앞의 내용이 잘 기억나지 않아 앞의 관련강의를 다시 찾아 봤습니다.

embedding layer가 자연어 처리에서 토큰을 numeric하게 바꿔주는 역할을 한다는 것을 배웠습니다.

그런데 상황에 따라 embeding layer을 학습하기도 한다는 것이 무슨 의미인지 잘 이해가 가지 않습니다.

2020/7/25

모두를 위한 딥러닝2 44, 45, 46, 47, 48를 학습했습니다.

잘 이해가 안가는 용어나 원리는 구글에 검색하여 사전과 다른 설명을 참고했습니다.

어제 1시간 30분 학습하였고 오늘 2시간 30분 총 4시간 학습하였습니다.

RNN을 여러층으로 활용하는 것을 stacking이라고 부른다는 것을 알았습니다.

dropout을 건다는 것이 무슨뜻인지 잘 이해가 안가 찾아봤습니다.

LSTM GRU가 무엇인지 찾아보도록 하겠습니다.

각 sequence에 대하여 출력을 내는 many to many 방식이 무엇인지 배웠습니다.

masking이라는 것이 pad토큰에 대해서는 loss를 계산하지 않는 다는 것임을 알게 되었습니다

sequence 순서대로 진행되면 forward rnn, 그 반대면 backward rnn이라는 것을 배웠습니다.

RNN에 활용되는 encoder와 decoder에 대하여 배웠습니다.

RNN 학습중 tearcher forcing이 무엇인지에 대하여 배웠습니다.

중요한 핵심 내용에 집중하는 attention이 어떤 것인지 알게되었습니다.

2020/7/27

파이썬 라이브러리를 활용한 머신러닝책의 1장을 모두 나갔습니다. 용어 설명은 복습한다는 생각으로 보았고 실습은 아나콘다로 진행하였습니다.

학습시간은 약 2시간 학습 30분정도 학습하였습니다.

과거 지능형 애플리케이션을 만들 때는 규칙 기반 전문가 시스템을 사용했으나 단점이 존재하였고, 이 때문에 머신러닝이 등장했다는 사실을 알게 되었습니다.

비지도 학습과 지도 학습의 명확한 정의와 그것들의 다양한 예시들에 대하여 알게 되었습니다.

샘플, 데이터 포인트, 특성, 특성 추출, 특성 공학 이라는 용어들이 무엇인지 알게 되었습니다.

머신러닝에서 파이썬이 이용되는 이유를 파이썬의 특징들을 중심으로 이해하였습니다.

과학 계산용 함수를 모아놓은 파이썬 패키지 Scipy에 대하여 알게되었습니다.

또한 붓꽃의 데이터를 가지고 붓꽃의 품종을 예측하는 실습을 진행하였습니다.

2020/7/28

파이썬 라이브러리를 활용한 머신러닝 2장의 앞부분을 학습하였습니다.(53p ~ 78p)

2시간 학습

지도학습과 관련된 용어들 (이진 분류, 다중분류, 회귀, 과대적합, 과소적합, 일반화) 와 같은 용어를 복습했습니다.

모델 복잡도와 데이터셋 크기의 상관관계에 대하여 알게 되었습니다.

훈련 데이터셋을 저장하는 방식으로 모델을 만들고, 새로운 데이터 포인트에 대하여 예측할 때는 최근접 이웃을 찾는 알고리즙인

k-NN (K 최근접 이웃) 알고리즘에 대하여 알게 되었습니다.

이웃의 수를 다르게 하여 훈련한 뒤 테스트 정확도를 측정하는 실습을 진행하였습니다.

kneighbors 분류기에서 데이터 포인트 사이의 거리를 잴때 유클라디안 거리 방식을 주로 사용함을 알았습니다.

k-NN은 훈련 세트가 매우 크면 예측이 느려지며, 많은 특성을 가지거나 특성값의 대부분이 0인 데이터 셋에서는 잘 작동하지 않는다는 단점을 가지고 있음을 알게되었습니다.

평균제곱오차를 최소화하는 파라미터 w와 b를 찾는 최소제곱법(선형 회귀)에 대하여 배웠습니다.

2020/7/29

파이썬 라이브러리를 활용한 머신러닝의 78p ~ 85p를 학습했습니다. 약 한시간 가량 진행하였습니다.

가중치의 절댓값을 작게 만드는 작업을 하는 리지 회귀에 대하여 배웠습니다.

alpha 값이 줄어들면 그만큼 계수에 대한 제약이 풀림을 알았습니다.

데이터 셋의 크기가 커지면 커질수록 선형회귀와 리지의 차이가 줄어듦을 배웠습니다.(데이터가 많을수록 규제항의 중요도는 떨어짐)

계수를 0으로 만들어버리는 L1규제를 하는 라소(Lasso)에 대하여 배웠습니다.

라소, 리지 회귀를 책을 참고하여 실습해 보았습니다.

2020/7/30

* 파이썬 라이브러리를 이용한 머신러닝 86p ~ 1시간 40분 가량 학습했습니다.
* 이진 분류가 무엇인지와 이진 분류 시 예측을 위한 방정식과 선형모델에서의 결정 경계에 대하여 알게 되었습니다.
* 알고리즘들이 만드는 잘못된 분류의 수를 최소화하도록 w와 b를 조정하는 것은 불가능하다는 것을 알았습니다.
* 선형 분류 알고리즘인 로지스틱 회귀와 서포트 벡터 머신에 대하여 학습했습니다.
* 두 모델은 기본적으로 L2규제를 사용합니다.
* 두 모델 모두 매개변수 C를 통해 규제의 강도를 결정합니다. C값이 커지면 규제가 감소하며, 개개의 데이터 포인트에 대하여 정확히 분류하고자 하게 됩니다. (즉 C가 너무 크면 과대 적합될 수 있습니다.)
* solver의 기본값이 lbfgs로 되어있으므로 L1 규제를 사용하려면 liblinear로 지정해야함을 알았습니다.
* 선형 분류 모델을 태생적으로 다중 클래스를 지원하지 않으므로 일대다 방법을 통해 다중 클래스를 지원합니다.
* 클래스의 수만큼 이진 분류 모델을 만들어 가장 결과값이 높은 클래스를 예측 결과로 합니다.
* 클래스들의 경계로 인하여 생긴 영역에서는(모두 속하는) 분류 공식의 결과가 가장 높은 클래스, 즉 가장 가까운 직선의 클래스가 결과가 된다는 것을 알았습니다.
* 선형 회귀 모델에서 중요한 특성이 많지 않은 경우 또는 모델의 해석이 중요한 요소일 경우 L1규제를, 그렇지 않으면 기본적으로 L2규제를 사용한다는 것을 배웠습니다.
* 선형모델의 장점에 대하여 알게 되었습니다.
* 선형모델은 학습 속도와 예측이 빠르며, 큰 데이터셋과 희소한 데이터셋에서도 잘 작동합니다. (대용량 데이터셋의 경우 solver = sag 옵션을 통해 빠르게 데이터를 처리하게 하거나, SGDClassifier와 SGDRegressor을 통해 대용량 처리를 할 수 있다는 것을 알았습니다.) 또한 선형 모델은 예측이 어떻게 만들어지는지 비교적 쉽게 이해할 수 있고, 샘플에 비해 특성이 많을 때 잘 작동한다는 것을 알았습니다.
* 나이브 베이즈 분류기에 대해서도 학습하였습니다.
* 나이브 베이즈 분류기는 각 특성을 개별로 취급하여 파라미터를 학습시키고, 각 특성에서 클래스별 통계를 단순하게 취합하여 효과적으로 분류를 합니다. GaussianNB는 연속적인 어떤 데이터에도 적용할 수 있고, BernoulliNB는 이진 데이터를, MultinominalB는 카운트 데이터에 적용된다는 것을 알았습니다.

나이브 베이즈 모델은 훈련과 예측 속도가 빠르고 훈련 과정을 이해하기 쉬우며, 희소한 고차원 데이터에서 잘 작동하며, 비교적 매개변수에 민감하지 않다는 장점이 있다는 것을 알았습니다.

2020/8/1

* 파이썬 라이브러리를 활용한 머신러닝 101p ~ 133p을 학습하였으며 3시간 40분 가량 학습했습니다
* 결정트리, 앙상블에 대하여 학습함
* 루트 노드에서 리프 노드까지 엣지로 질문의 답과 다음 질문을 연결한 형태
* 결정트리를 지도 학습 방식으로 데이터로부터 학습할 수 있음
* 결정트리에서 학습한다는 것은 정답에 가장 빨리 도달하는 질문 목록을 학습한다는 것 (이런 질문을 테스트라고 함)
* 트리를 만드는 알고리즘은 모든 테스트에서 타깃값에 대해 가장 많은 정보를 가진 것을 고르는 식으로 진행함
* 테스트는 하나의 특성에 대해서만 이루어지기 때문에 항상 축에 평행
* 테스트는 한 개의 타깃값을 가질 때까지 반복함.
* 타깃 하나로만 이뤄진 리프 노드를 순수 노드 라고 함
* 새로운 데이터 포인트에 대한 예측은 데이터 포인트가 분할된 특성들의 영역중 어디에 놓이는지 확인하면 됨 (루트 노드를 시작으로 테스트 결과에 따라 하위 트리를 탐색해나가는 식으로 영역을 찾음)
* 회귀 문제에서도 트리를 사용하는 것이 가능함
* 테스트 결과에 따라 트리를 탐색해 나가고 새로운 데이터 포인트에 해당하는 리프 노드를 찾는 방식으로 예측함 이때 훈련 데이터 평균값이 이 데이터 포인트의 출력이 됨
* 트리 만들기를 할때 모든 리프 노드가 순수 노드가 될 때까지 진행하면 모델이 복잡해지고 데이터가 과대적합될 수 있음
* 트리의 최대 깊이나 리프의 최대 개수를 제한하는 사전 가지치기방식 또는 사후 가지치기(가지치기) 방식으로 과대적합을 막을 수 있음.
* scikit-learn 에서 결정 트리는 DecisionTreeRegressor 와 DecisionTreeClassifier에 구현되어 있으며 사전 가지치기만 지원함.
* 트리를 만드는 결정에 각 특성이 얼마나 중요한지를 평가하기 위해 특성 중요도 라는 것을 사용함. (0과 1사이의 수로 1에 가까울수록 완벽하게 예측했다는 뜻, 특성 중요도의 합은 1)
* 특성 중요도가 낮다고 해서 그 특성이 유용하지 않다는 것이 아니라 단지 트리가 그 특성을 선택하지 않았을 뿐이며 다른 특성이 동일한 정보를 지니고 있어서일 수 있음
* DecisionTreeRegressor는 훈련 데이터의 범위 밖의 포인트에 대한 예측을 할 수 없음
* 결정 트리는 모델을 쉽게 시각화 할 수 있으며, 데이터의 스케일에 구압ㄷ지 않는 장점이 있음 (특성의 정규화나 표준화 같은 전처리 과정이 필요 없음) 일반화 성능이 좋지은 않은 단점이 있음
* 여러 머신러닝 모델을 연결하여 새로운 모델을 만드는 기법을 앙상블이라고 함.
* 랜덤 포레스트는 여러 결정 트리의 묶음임
* 서로 다른 방향으로 과대적합된 트리를 여러개 만들어 그 결과를 평균냄으로써 과대적합된 양을 줄이는 방식임
* 랜덤 포레스트에서 트리를 만들 때, 데이터 포인트를 무작위로 선택하는 방법과 분할 테스트에서 특성을 무작위로 선택하는 방법으로
* 트리를 만듦
* 랜덤 포레스트를 만들 때는 우선 생성할 트리의 개수를 먼저 정해야함
* 트리를 만들기 위해서는 데이터의 부트스트랩 샘플을 생성해야함 (데이터 포인트 중에서 무작위로 데이터를 n번 반복 추출함)
* 랜덤 포레스트로 예측할 때에는 우선 모델에 있는 모든 트리의 예측을 만듦 (회귀에서는 예측들의 평균을 최종 예측으로 함)
* 이후 약한 투표 전략을 사용하여 분류함
* 각 알고리즘이 가능성 있는 출력 레이블의 확률을 제공하여 간접적인 예측을 하고 예측한 확률을 평균내어 가장 높은 확률을 가진 클래스를 예측 값으로 함.
* 랜덤 포레스트는 매개변수 튜닝을 많이 하지 않아도 되며 데이터의 스케일을 맞출 필요도 없다는 장점이 있음
* 단일 트리의 장점은 가져오고 단점은 보완함
* 대신 트리가 깊어지며 분석하기가 어려움
* 랜덤 포레스트는 random\_state에 따라 전혀 다른 모델이 만들어짐 (트리가 많을 수록 random\_state 값의 변화에 따른 변동이 적음)
* 텍스트 데이터와 같이 매우 차원이 높고 희소한 데이터에서는 잘 작동하지 않음 (선형모델이 더 적합) 대신 아주 큰 데이터 셋에 비교적 잘 작동하고 여러 cpu 코어로 간단하게 병렬화가 가능함 (대신 메모리를 많이 먹으며 훈련과 예측이 느림)
* 매개변수 max\_feature을 통해 트리가 얼마나 무작위가 될지 결정함
* 분류는 max\_feature=sqrt, 회귀는 max\_feature=n\_features
* 그레이디언트 부스팅 회귀 트리는 분류와 회귀 모두 사용 가능
* 이전 트리의 오차를 보완하는 방식으로 순차적으로 트리를 만듦 (무작위성이 없고 강력한 사전 가지치기를 사용함)
* 보통 하나에서 다섯 정도의 트리를 사용 ( 메모리를 적게 사용, 예측도 빠름) - 약한 학습기 라고도 함
* 각각의 트리는 데이터의 일부에 대하서만 예측을 수행하므로 트리가 많이 추가될수록 성능이 좋아짐.
* 매개변수를 잘 조정해야 하며 훈련시간이 길다는 단점이 있음.
* 대신 특성의 스케일을 조정하지 않아도되며 이진 특성과 연속적인 특성에도 동작함. 다만 희소한 고차원 데이터에서는 잘 작동하지 않음.
* n\_estimate와 learning\_rate로 이전 트리의 오차를 보정함 (n\_estimations 값을 키우면 트리가 추가됨, 과대적합될 확률이 올라감)
* 앙상블 알고리즘인 배깅은 Bootstrap agregating의 줄임말로 중복을 허용한 랜덤 샘플링으로 만든 훈련 세트로 분류기를 각기 다르게 학습시키는 알고리즘
* 확률값을 평균하여 예측을 수행하며 분류기가 predict\_proba()를 지원하지 않으면 빈도 높은 클래스를 예측 결과로 함
* 엑스트라 트리는 후보 특성을 무작위로 분할한 다음 최적의 분할을 찾는 알고리즘
* DecisionTreeClassifier(splitter='random')을 사용하고 부트스트랩 샘플링은 적용하지 않음
* 무작위성이 증가되면 일반적으로 모델의 편향이 늘어나고 분산이 감소함
* 예측 방식은 랜덤 포레스트와 동일함
* 에이다 부수트는 그레이디언트 부스팅과 같은 약한 학습기를 사용함 ( 다만 잘못 분류한 샘플에 가중치를 높여서 다음 모델을 훈련시킴)

선능에 따라 훈련된 각 모델에 가중치가 부여되며 모델이 예측한 레이블을 기준으로 모델의 가중치를 합산하여 가장 높은 값을 가진 레이블을 선택하여 예측결과로 함

2020/8/3

* 174p 까지 하여 지도학습 마무리 했으며, 23p에서 52p 까지 복습하였습니다. 2시간 30분 정도 학습 하였습니다.
* Decision\_function과 predict\_proba 메서드를 통해 다중분류에서도 불확실성을 측정할 수 있음
* 다중 분류에서 decision\_function의 결괏값의 크기는 (n\_samples, n\_classes ) 임
* 각 열은 클래스에 대한 확신 점수를 담고 있음
* 열이 n\_class 이면 열을 가로질러서 argmax 함수를 적용해 예측 결과를 재현할 수 있음
* 클래스가 문자열이거나 정수형을 사용하지만 연속적이지 않고 0부터 시작하지 않을 수 있음

분류기의 classes\_ 속성을 사용해 클래스의 실제 이름을 얻어야 함

2020/8/5

* 파이썬 라이브러리를 이용한 머신러닝 175p ~ 202p 4시간 가량 학습했습니다.
* 교재가 생각보다 늦어서 비지도학습 부분을 먼저 들어갔습니다. 코드 실습 위주로 진행했습니다.
* 비지도 학습에는 비지도 변환과 군집이 있음
* 비지도 변환 : 데이터를 새롭게 표현하여 사람이나 다른 머신러닝 알고리즘이 원래 데이터보다 쉽게 해석할 수 잇도록 만드는 알고리즘
* 고차원 데이터를 특성의 수를 줄이면서 꼭 필요한 특징을 포함한 데이터로 표현하는 방법인 차원 축소에서 널리 사용됨.
* 군집 알고리즘 : 데이터를 비슷한 것끼리 그룹으로 묶는 알고리즘
* 비지도 학습은 레이블이 없는 데이터를 사용하기 때문에 학습한 것이 유용한지 평가하는 것이 어려움
* 탐색적 분석 단계 또는 지도 학습의 전처리 단계에서도 주로 사용함
*  지도 학습의 정확도가 좋아지고 메모리와 시간을 절약할 수 있음
* 데이터 전처리
* StandardScaler : 특성의 평균을 0, 분산을 1로 변경하여 모든 특성이 같은 크기를 가지게 함, 특성의 최솟값과 최댓값의 크기를 제한하지는 않음
* RobustScaler : StandardSclaler와 유사하나 이상치에 영향을 받지 않게 함
* MinMaxScaler : 모든 특성이 정확하게 0과 1 사이에 위치하도록 전처리함
* Normalizer : 특성 벡터의 유클리디안 길이가 1이 되도록 데이터 포인트를 조정함 (반지름이 1인 구 위에 데이터 포인트를 놓는 식)특성 벡터의 길이는 상관 없고 데이터의 방향만이 중요할 때 주로 사용
* QuantileTransformer : 1000개의 분위를 사용하여 데이터를 균등하게 분포시킴
* 이상치에 민감하지 않으며 전체 데이터를 0과 1사이로 압축함
* PowerTransformer : 데이터의 특성별로 정규분포 형태에 가깝도록 변환해줌
* 일반적으로 시각화, 데이터 압축, 추가적 처리 (주로 지도 학습의 데이터로 사용하기 위해)를 위해 정보가 더 잘 드러나는 표현을 찾기 위해 데이터를 비지도 학습으로 변환함
* 주성분 분석 (PCA) : 특성들이 통계적으로 상관관계가 없도록 데이터셋을 회전시키는 기술
* 1. 분산이 가장 큰 방향을 찾음
* 2. 그 방향과 직각인 방향중에서 가장 많은 정보를 담은 방향을 찾음
* 3. 축에 나란하도록 회전함
* 4. 데이터에서 평균을 빼서 중심을 원점에 맞춤
* 5. 데이터에 다시 평균을 더하고 반대로 회전시킴

주로 노이즈를 제거하거나 주성분에서 유지되는 정보를 시각화 할 때 사용함

2020/8/6

* 파이썬 라이브러리를 활용한 머신러닝 202p ~ 225p 복습 포함 4시간 20분 학습하였습니다. 딥러닝 책이 와서 이제부터 그 책으로 진도를 나갈 것 같습니다.
* 특성 추출은 원본 데이터 표현보다 분석하기에 더 적합한 표현을 찾을 수 있을 것이라는 생각에서 출발함
* 이미지는 적색, 녹색, 청색의 강도가 기록된 픽셀로 구성
* PCA를 이용하여 LFW 데이터셋의 얼굴 이미지에서 특성을 추출하는 실습을 진행했음
* 얼굴 인식은 새로운 얼굴 이미지가 데이터베이스에 있는 기존 얼굴 중 하나에 속하는지 찾는 것
* 얼굴 데이터베이스에는 사람의 수가 많지만 각 사람에 대한 이미지는 적기 때문에 분류기를 훈련 시키기 어려움
* 또한 대규모 모델을 훈련시키지 않고도 새로운 사람의 얼굴을 추가할 수 있도록 구성해야 함
* 클래스마다 하나의 훈련 샘플을 사용하는 최근접 이웃 분류기를 사용하여 실습함(KNeighborsClassfier)
* 얼굴 인식을 최근접 이웃 분류기를 이용해 하려 하면 결과가 좋지 않음
* 그 이유는 얼굴 유사도 측정할때 픽셀 공간에서 거리를 계산하는 것으로 하기 때문임
* -> 각 픽셀의 회색톤 값을 다른 이미지에서 동일한 위치에 있는 픽셀 값과 비교하는 식으로 진행
* -> 사람이 얼굴 이미지를 확인하는 방식과 많이 다르고 얼굴의 특징을 잡아내기 힘듦
* -> PCA 화이트닝 옵션을 사용하면 정확도를 올릴 수 있음
* PCA변환을 활용 할때 주성분의 개수를 늘리면 이미지가 더욱 상세해짐
* 주성분을 픽셀 수만큼 사용할 시 변환 후에 어떤 정보도 잃지 않게 되어 이미지를 완벽하게 재구성 할 수 있음
* 비음수 행렬 분해(NMF)
* NMF는 유용한 특서을 뽑아내기 위한 비지도 학습 알고리즘
* PCA와 유사하고 차원 축소에도 사용 가능
* 어떤 성분의 가중치 합으로 각 데이터 포인트를 표현 할 수 있음
* PCA와 달리 음수가 아닌 성분과 계수 값을 찾음 (주성분의 계수가 모두 0이상)
* 오디오 트랙이나 여러 악기로 이뤄진 음악과 같이 독립적 소스를 추가하여 만들어진 데이터에 유용함
* 데이터가 원점을 기준으로 상대적으로 어디에 놓여있는지가 NMF에서 중요함
* 원점에서 데이터로 가는 방향을 추출한 것으로 음수 미포함 성분을 이해할 수 있음
* 데이터를 완벽하게 재구성할 수 있을 만큼 성분이 많으면 알고리즘은 데이터의 각 특성의 끝에 위치한 포인트를 가리키는 방향을 선택함
* PCA와 달리 성분 개수를 줄이면 특정 방향이 제거되는 것 뿐만 아니라 전체 성분이 완전히 바뀜
* 또한 모든 성분은 동등하게 취급됨
* 매니폴드 학습 알고리즘은 복잡한 매핑을 만들어 더 나은 시각화를 제공함 주로 t-SNE 알고리즘을 많이 사용함 (목적이 시각화)
* 시각화를 위한 알고리즘이므로 3개 이상의 특성을 뽑는 경우는 거의 없음
* 테스트 세트에는 적용할 수 없고 훈련했던 데이터만 변환함
* 탐색적 데이터 분석에는 유용하지만 지도 학습용으로는 거의 사용하지 않음
* t-SNE의 아이디어는 데이터 포인트 사이의 거리를 가장 잘 보존하는 2차원 표현을 찾는 것임
* 1. 각 데이터 포인트를 2차원에 무작위로 표현함
* 2. 원본 특성 공간에서 가까운 포인트는 가깝게, 멀리 떨어진 포인트는 멀어지게 만듬

-멀리 떨어진 포인트와 거리를 보존하는 것보다 가까이 있는 포인트에 더 많은 비중을 둠 (이웃 포인트에 대한 정보 보존)

2020/8/7

* 텐서플로 2.0 프로그래밍 1p ~ 34p 텐서플로우에 대하여 공부하고 설치하고 실행해보는 실습을 진행했씁니다.
* 또한 파이썬 라이브러를 활용한 머신러닝 101p 까지 복습했습니다. 총 3시간 학습했습니다.
* 텐서플로 : 머신러닝을 위한 오픈소스 플랫폼
* 구글이 주도적으로 개발하는 현재 가장 널리 쓰이는 딥러닝 프레임 워크
* 구글 코랩으로 코드를 자유롭게 공유 가능
* 구글 클라우드는 빅 쿼리 AutoML 등 텐서플로를 이용해 효과적인 작업 처리 기술 세트를 제공
* 텐서플로는 파이썬과 C++ API를 기본적으로 제공하고 자바스크립트, 자바, 고, 스위프트 등과 같은 다양한 프로그래밍 언어 지원
* 텐서플로 2.0 되면서 변화 한점
* 1. API 정리
* 2. 즉시 실행 모드 -> 그래프를 생성하지 않아도 텐서를 만들고 값을 계산
* 3. 세션 대신 함수 -> tf.Session 대신 함수 앞에 @tf.function을 붙여서 일반 연산을 그래프 연산으로 변환 (코드 간결화)
* 4. tf.keras (tf.contrib는 삭제)
* - 고수준 API
* - 신경망 기초 이론을 잘 몰라도 복잡하고 다양한 신경망 레이어를 쉽게 생성하고 학습시킬 수 있음
* 5. TPU 지원
* - 구글에서 딥러닝을 위해 설계한 하드웨어, 구글 클라우드에서 사용가능, 구글 코랩에서도 런타임 유형을 바꿔 사용 가능
* - GPU보다 전력을 적게 소모하여 경제적이고 친환경적

- 32비트로 수행되는 곱셈 연산을 최적화된 16비트로 낮춰 연성 수행 (계산 속도 빠름)

2020/8/8

* 시작하세요 텐서플로우 2.0 프로그래밍 34p ~ 67p 약 3시간 20분 학습했습니다.
* %기호 : (매직 커맨드) 인터랙티브 파이썬부터 지원하는 명령, 여러가지 기능을 미리 정의해놓고 편리하게 사용할 수 있게 해주는 역할
* - %tensorflow\_ersion 2.x : 텐서플로우 버젼 선택, 구글 코랩에서만 지원됨
* tf.\_\_version\_\_ 으로 버젼 확인
* 퍼셉트론 이라는 책에서는 AND, OR, XOR 연산을 예시로 인공 신경망의 원조격인 퍼셉트론의 한계를 지적했음
* 초기화 : 처음에 신경망의 초깃값을 지정해주는 것
* = Xavier 초기화 또는 He 초기화 등이 있음
* tf.random.uniform 함수를 통해 균일 분포의 난수를 얻을 수 있음
* 균일분포 : 최솟값과 최댓값 사이의 모든 수가 나올 확률이 동일한 분포에서 수를 뽑는 다는 뜻
* tf.Tensor 에서 shape는 행렬을 구성하는 행, 열 등 차원의 수를 나타냄
* shape의 두번째와 세번재 인수가 0,1이면 이 값은 최솟값과 최댓값을 의미 (이 사이의 난수를 뽑음)
* dtype은 자료형을 의미 float32는 32비트 부동소수점 수라는 뜻 (float64, float16 도 있음)
* tf.random.normal 일경우 두번째 숫자는 정규분포의 평균을 세번재는 표준편차를 의미함
* 뉴런은 입력을 받아서 계산 후 출력을 반환하는 단순한 구조
* 신경망은 뉴런 여러개가 모인 레이어가 여럿 모여 구성됨
* 뉴런은 입력, 가중치, 활성화 함수, 출력으로 구성
* 활성화 함수 : 뉴런의 출력값을 정하는 함수
* - 주로 시그모이드 ReLU 등을 주로 씀 딥러닝에는 ReLU가 더 자주 쓰임
* - 시그모이드에는 오류를 역전파 할때 값을 점점 작아지게 하는 문제가 있음
* 가중치 : 학습할 때 변화하는 값, 초기화를 통해 랜덤한 값을 얻은 후, 학습과정에서 일정한 값으로 수렴
* 경사 하강법 : 가중치에 입력과 학습률과 에러르 곱한 값을 더해 학습하는 방법
* 편향(bias) : 가중치에 변화가 없어 뉴런이 학습되지 못하는 상황을 방지하기 위해 넣는 값
* numpy : 수학과 과학 연산에 특화된 파이썬 모듈
* array : numpy에서 제공하는 자료형, 리스트보다 적은 메모리를 차지하고 계산 속도가 빠름
* 파이썬 리스트에 어떤 값을 곱하면..
* -> 양의 정수이면 그 만큼 반복된 리스트 반환
* -> 0이하의 수는 빈 리스트 반환
* -> 정수가 아닌 수의 경우 에러가 발생
* numpy의 array에 어떤 값을 곱하면..
* -> array에 각 원소에 대해 자동으로 실수를 곱하는 연산이 이뤄짐
* 하나의 퍼셉트론으로 xor문제를 해결할 수 없기 때문에 여러개의 퍼셉트론을 사용해야됨
* 3개의 퍼셉트론으로 해결하는 실습을 진행함
* tf.keras에는 추상적인 클래스인 model이 있음
* 딥러닝 계산을 위한 여러 함수와 변수의 묶음임, 주로 tf.keras.Sequential 구조를 많이 씀
* 시퀀셜 모델의 인수로는 레이어가 차례대로 정의된 리스트를 전달함
* tf.keras.layer.Dense를 통해 model에서 사용하는 레이어를 정의함
* Dense는 입력과 출력 사이에 있는 모든 뉴런이 서로 연결되는 가장 기본 레이어
* tf.keras.layers.Dense 안의 units는 레이어를 구성하는 뉴런의 수를 정의함
* 뉴런이 많으면 레이어 성능은 올라가지만 계산량이 늘어나고 메모리가 많이 차지됨
* Dense 레이어의 파라미터 수는 (입력측 뉴런의 수+1 ) \* (출력측 뉴런의 수)로 계산
* model.fit으로 epochs만큼 학습시키고

model.predict으로 성능을 평가

2020/8/9

* 시작하세요 텐서플로우 2.0 프로그래밍 67p ~ 103p 까지 교재의 실습 위주로 학습했습니다.
* 3시간 가량 학습했습니다.
* 시각화를 위해서 matplotlib.pyplot을 이용함
* plt.plot(x,y) : x축 y축를 넣어 그래프 생성
* - 기본은 꺾은선 그래프
* plot() 안의 매개 변수
* 'bo' : 파란 점
* 'b-' : 선
* 'b--' : 점선
* b대신 r(빨간색), y(노란색), g(초록색) 등으로 바꿀 수 있음
* plt.xlabel(x축 제목), plt.ylabel(y축 제목) : 축 제목 설정
* plt.show() : 그래프 확인
* plt.hist() : 히스토그램만들기
* 선형 회귀 : 데이터의 경향성을 가장 잘 설명하는 하나의 직선을 예측하는 것
* 두 개 이상의 리스트를 하나로 묶는 기법 : list(zip(list\_1, list\_2))
* 랜덤한 값으로 변수 초기화 : a = tf.Variable(random.random())
* 잔차 : 기대출력에서 실제출력을 뺀 것
* loss : 잔차의 제곱을 모두 더하여 평균을 낸 값
* optimizer : 손실을 최소화하는 알고리즘을 자동으로 진행해주는 함수(도구)
* ex) tf.optimizers.Adam(lr = 0.07) => 최적화 함수 adam 을 학습률을 0.07로
* optimizer.minimize(copute\_loss, var\_list = [a,b] ) : 잔차의 제곱의 평균을 최소화
* 다항회귀 : 회귀선이 직선 대신 2차 3차 함수등의 곡선이 되는 회귀
* 딥러닝 에서는 데이터를 전처리해서 정규화 해야 학습 효과가 좋음
* 정규화 : 데이터에서 평균값을 뺀 다음 표준편차로 나누는 것
* model.fit()에서 validation\_split 인수 : 검증 데이터로 떼서 학습 결과를 검증할 데이터의 퍼센트를 결정함
* callback함수 : 학습 도중 끼어들기 위해 사용, 모델을 학습할 때 에포크가 끝날 때마다 호출됨. model.fit() 함수에 callback 인수를 사용해 콜백 함수의 리스트를 지정할 수 있음

tf.keras.callbacks.EarlyStopping : 콜백 함수의 리스트에 들어가 학습을 일찍 멈추는 기능을 하는 함수

2020/8/11

* 시작하세요 텐서플로우 2.0 프로그래밍 103p ~ 123 약 2시간 30분 학습했습니다.
* 와인 데이터셋을 불러와 이항분류 하는 실습을 진행했습니다.
* 분류 : 데이터가 어느 범주에 해당하는지 판단하여 유사한 것들을 같은 범주로 묶으며 학습
* 분류는 퍼센트 수치로 정확도를 측정
* 분류는 명확한 정답이 없는 경우도 있음. - 비지도 학습
* 이항분류 : 정답의 범주가 두 개인 범주 문제
* pandas 라이브러리 : 데이터 정제 및 분석을 위한 라이브러리 (pd로 축약하여 사용)
* pd.read\_csv() : csv파일을 pandas의 데이터 구조인 데이터프레임으로 읽어옴
* 데이터 프레임 : 행(index) 열(attribute)로 구성된 데이터를 다루는 데이터 구조
* head(숫자) : 데이터프레임의 앞에서부터 원하는 갯수의 행 출력
* tail(숫자) : 데이터프레임의 뒤에서부터 원하는 갯수의 행 출력
* 데이터 프레임에 새로운 속성 추가하기 :
* 딕셔너리 타입처럼 red['type'] = 0 이런식으로 하면 됨
* pd.concat() : 데이터 프레임을 위아래로 연결시켜 새로운 데이터 프레임을 만듦
* describe() : 데이터프레임의 간단한 통계 정보를 확인
* value\_counts() : 어떤 속성에 존재하는 각 값의 수를 출력함
* 데이터를 훈련 데이터와 테스트 데이터로 나눌때 데이터를 랜덤하게 섞어서 함
* -> 한 범주의 데이터 양이 매우 많거나 적으면 일정한 비율의 데이터가 훈련 데이터와 테스트 데이터에 모두 들어갈 수 있도록 처리가 필요
* 판다스의 info()함수 : 데이터 프레임을 구성하는 속성들의 정보를 알려줌 (정규화 과정에서 데이터에 숫자가 아닌 값이 있으면 에러남)
* -외부에서 불러오는 데이터는 어떤 값으로 구성돼 있는지 알기 어려울 수 있음 이때 데이터 파악을 도와주는 함수
* 판다스의 sample() : 데이터 프레임에서 frac 인수로 지정된 비율만큼 행을 랜덤하게 뽑아 새로운 데이터프레임을 만듦
* to\_numpy() : 데이터프레임을 넘파이 array로 변환함
* tf.keras.utils 에서 불러오는 to\_categorical : 정답 행렬을 원-핫 인코딩 방식으로 바꿈
* 원-핫 인코딩 방식 : 정답에 해당하는 인덱스의 값에는 1 나머지에는 0을 넣는 방식
* 소프트 맥스 함수 : 출력 값들을 로그의 밑인 e의 지수로 사용해 계산하여 모두 더한 값으로 나누는 함수
* - 분류 문제나 언어 RNN에서 다음 토큰 예측, 강화학습에서 에이전트 행동 확률 구하기 등 결괏값으로 확률이 필요한 다양한 분야에 사용

model.evaluate() 함수 : 모델의 성능 평가

2020/8/13

* 시작하세요 텐서플로우 2.0 프로그래밍 123p ~ 173p 4시간 40분 학습하였습니다.
* 와인의 품질 데이터와 Fashion MNIST 데이터를 가지고 실습하였습니다
* 다항분류 : 범주의 수가 2개를 초과하는 분류
* 데이터 프레임 loc 함수 : 특정한 데이터의 인덱스를 골라내는 함수
* -대괄호 안 인수를 하나만 넣으면 행을 골라내고, 쉼표를 포함한 두 개의 인수를 넣으면 차례대로 행, 열을 골라냄
* Sparse : 희소 행렬
* sparse\_categorical\_crossentropy : 데이터 전처리 없이 희소 행렬을 나타내는 데이터를 정답 행렬로 사용할 수 있게 함
* Flatten 레이어 : 다차원 데이터를 1차원으로 정렬하는 역할
* 특징 추출 기법에 대하여 학습함
* 이미지 데이터에서는 연구자가 스스로 특징을 찾아야 함
* 이미지에서 사물의 외곽선에 해당하는 특징을 찾아 물체 감지 알고리즘을 만들 수 있음
* SIFT 알고리즘 : 이미지의 회전과 크기에 대해 변하지 않는 특징을 추출해 두개의 이미지에서 서로 대응되는 부분을 찾아냄
* 컨볼루션 연산 (합성곱) : 각 픽셀을 본래 픽셀과 그 주변 픽셀의 조합으로 대체하는 동작
* - 이떄 쓰이는 작은 행렬을 필터 또는 커널이라고 함
* - 원본 이미지의 각 픽셀을 포함한 주변 픽셀과 필터의 모든 픽셀은 각각 곱연산, 그 결과를 모두 합해 새로운 이미지를 넣어줌
* - 각 필터의 생김새에 따라 수직선/수평선 검출, 흐림 효과, 날카로운 이미 효과 등을 줄 수 있음
* 보통의 필터들은 경험적 지식을 통해 직접 손으로 값을 넣어줌 - 수작업으로 설계한 특징 이라고 함
* 적용하고자 하는 분야에 대한 전문적 지식이 필요하며, 시간과 비용이 많이 들며, 한분야에서 효과적인 특징을 다른 분야에 적용시키 어렵다는 단점이 있다.
* 딥러닝 기반 컨볼루션 연산은 네트워크가 특징을 추출하는 필터를 자동으로 생성하여 위 문제들이 해결됨
* Dense 레이어 : 가장 기본이 되는 레이어, 각 뉴런이 서로 완전히 연결되어 완전 연결 레이어라고도 불림
* 분류를 위한 일반적인 컨볼루션 신경망은 특징 추출기와 분류기가 합쳐져 있는 형태
* 컨볼루션 레이어와 풀링 레이어가 특징 추출기 역할을, Dense 레이어가 분류기 역할을 함
* 특징 추출기에는 컨볼루션 레이어와 풀링 레이어가 교차되어 배치됨
* 분류기에는 Dense 레이어가 배치됨, 과적합을 막기 위해 드롭아웃 레이어가 Dense 레이어 사이에 배치됨 (마지막 Dense레이어 뒤는 제외)
* 컨볼루션 레이어 : 컨볼루션 연산을 하는 레이어 - 필터는 학습을 통해 자동으로 추출, 코드로 필터의 개수를 지정함
* -이미지에는 원색으로 구성된 각 이미지가 가진 색상에 대한 정보를 분리해서 담아놓은 공간인 채널이 있음
* -각 채널에 대해 계산된 값을 합쳐 새로운 이미지를 만들어냄
* -새로운 이미지의 마지막 차원 수는 필터의 수와 동일
* -다른 레이어와 마찬가지로 tf.keras.layers에서 임포트
* Conv2D 레이어를 생성할 때 인수는 kernel\_size, strides, padding, filters 총 4개임
* kernel\_size : 필터 행렬의 크기 (수용 영역)
* strides : 계산 과정에서 한 스텝마다 이동하는 크기
* padding : 컨볼루션 연산 이전 입력 이미지 주변에 빈 값을 넣을지 지정하는 옵션
* - valid : 빈 값 사용하지 않음
* - same : 빈 값을 넣어 출력 이미지의 크기를 입력과 같도록 보존 (빈값으로 0이 쓰일 경우 제로 패딩이라고 함)
* filter : 필터의 개수 많을 수록 좋지만 너무 많으면 학습 속도가 저하되고 과적합이 발생할 수 있음
* 이미지를 구성하는 인접 픽셀들이 비슷한 정보를 갖는 경우가 많음 -> 비효율적
* 중요한 정보만 남기기 위해 서브 샘플링 기법을 사용 -> 효율적 메모리 사용, 과적합 방지
* 이때 풀링 레이어를 사용함
* -Max 풀링 레이어, Average 풀링 레이어 등이 있음
* 사용 예시 ) tf.keras.layers.MaxPool2D(pool\_size=(2,2), strides=(2,2))
* 풀링 레이어에는 가중치가 존재하지 않아 학습되지 않음, 네트워크 구조에 따라 생략되기도 함
* 드롭아웃 레이어 : 네트워크 과적합을 막기 위해 만들어짐, 학습 과정에서 무작위로 뉴런의 부분집합을 제거함
* 생성 코드 ) tf.keras.layers.Dropout(rate=)
* -rate : 제외할 뉴런의 비율
* -가중치가 없어 학습되지 않음
* 복잡하고 파라미터가 많은 네트워크를 다룰때 구글 코랩에서 가속기를 쓰지 않으면 계산이 몹시 느려짐
* 가속기 사용 ) 런타임 -> 런타임 유형 변경 -> 하드웨어 가속기 -> GPU 지정
* 퍼포먼스를 높이기 위한 방법에는 '더 많은 레이어 쌓기' 와 '이미지 보강' 기법이 있음
* 더 깊은 중첩된 구조의 컨볼루션 레이어가 나올수록 퍼포먼스는 크게 개선되었음
* 이미지 보강 : 훈련 데이터에 없는 이미지를 새롭게 만들어 훈련 데이터를 보강하는 방법
* - 새로운 이미지는 훈련 데이터의 이미지를 원본으로 삼아 일정한 변형을 가해 만들어짐
* - 이미지를 가로로 뒤집고, 약간 회전시키거나, 기울이거나, 일부 확대하거나, 평행이동시켜서 다양한 이미지를 만들어냄 -> 표현력 향상
* 이미지 보강 작업은 tf.keras에 ImageDataGenerator 를 통해 할 수 있음
* rotation\_range, zoom\_range, shear\_range 등의 인수를 가짐
* 내장함수인 flow()함수로 이미지 생성

copy()함수로 복사본 안전하게 생성

2020/8/27

* 김기현의 자연어 처리 딥러닝 캠프 51p 까지 4시간 학습하였습니다.
* 아나콘다와 토치를 깔아 실습을 진행하였습니다.
* 자연어 처리 인공지능 : 사람의 언어를 컴퓨터가 알아듣도록 처리하는 인터페이스 -> 컴퓨터 공학과 언어학 지식 필요
* 자연어 처리 인공지능의 응용 분야
* 1) 감성 분석 ( 대량의 텍스트를 이해, 수치화)
* 2) 사용자 의도 파악 및 사용자와 대화하고 도움을 주는 봇
* 3) 요약, 기계번역과 같은 작업
* 4) 입력받은 값을 통해 검색 및 답변하는 기능
* 자연어 처리는 단어 간의 순서 및 상호 정보가 반영된 시퀀셜 데이터를 사용함
* 인공지능은 두번의 대유행과 두번의 빙하기가 있었음
* 두 번째 황금기였던 1980년대 역전파 알고리즘이 제안되어 기존 문제를 해결한 듯 했으나 곧 다시 침체기를 맞이함
* 2006년 여러 층의 은닉층을 효과적으로 사전훈련 시키는 DBN을 통해 침체기를 해결하나 했으나 그러지 못했음
* 2012년 이미지넷에서 인공 신경망을 이용한 AlexNet은 여러 층의 합성곱 계층을 쌓아 네트워크 구조를 만듦
* 2000년대 초창기 음성 인식은 GMM을 통해 음소를 인식하고 은닉 마르코프 모델을 통해 시퀀셜 정보를 반영해 학습하여
* 음향 모델과 n-gram 기반의 언어 모델을 WFST 방식을 통해 결합하는 전통적인 자동음성인식 시스템이었음
* 2012년 GMM을 DNN으로 대체하면서 큰 발전을 함 이후 음향모델을 LSTM으로 대체하고 end to end 방식이 성과를 내며 자리잡는 추세임
* 딥러닝 이전 기계번역은 통계 기반 기계번역을 사용하고 있었음
* 규칙 기반 기계번역 방식에 비해 언어 간 확장이 용이하며 성능도 뛰어나지만 구조가 매우 복잡했음
* 2014년 seq2seq라는 모델 구조가 등장하여 end to end 신경망 기반 기계 번역 시대가 오게 됨
* seq2seq를 기반으로 어텐션 메커니즘이 제안되며 신경 기반 기계번역으로 대통합이 이루어짐
* 판별 모델 학습 : 데이터 X가 주어지면 알맞은 레이블 Y를 찾아내는 것에 집중
* 생성 모델 학습 : 데이터 X의 분포 자체를 배우는 것에 집중
* 기존 자연어 처리 어플리케이션 구조는 여러 가지 단계의 모듈로 구성되어 디자인이 복잡하고 문제에 따라서 또 다른 서브 모듈이 추가 되기도 함
* 매우 무겁고 복잡하며 구현 및 시스템 구성이 어려움 또 각각의 모듈이 완벽히 동작할 수 없어 발생한 오차들이 중첩 및 가중되어 뒤로 전파되는 오파의 전파 현상등의 문제도 발생할 수 있음
* 전통적인 심볼릭 기반 접근 방법과 다른 딥러닝 기반 접근 방법의 특징
* 1. 연속적인 신경망 공간에서 이루어짐
* 2. 사람이 이해하기 어려움
* 3. 디버깅이 어려움
* 4. 연산 속도가 빠름
* 5. 모호성과 유의성에 강인함
* 6. end to end 모델을 통한 성능 개선과 시스템 간소화 가능
* 사람의 언어는 불연속적인 이산 심벌로 이루어져 있음 -> 불연속적 심벌로 데이터를 취급 -> 사람이 이해하고 해석하기 쉬움, 모호성이나 유의성을 다룰 때는 어려움
* word2vec 등의 단어 임배딩 -> 단어를 연속적인 벡터로 나타낼 수 있음 -> 직접적 해석은 어려워짐, 모호성과 유의성 문제를 더 잘 해결
* + end to end모델을 구현하여 더욱 높은 성능을 뽑을 수 있음
* 자연어 처리가 어려운 이유
* 1. 모호성
* 단어의 중의성, 문장 내 정보 부족등의 이유로 문제 발생
* 2. 다양한 표현
* 문장의 표현 형식은 다양하며 비슷한 의미의 단어들도 많아 같은 것을 묘사하는 문장도 다양하게 존재
* 사람들이 이해하는 의미가 미묘하게 다를 수도 있음
* 3. 불연속적 데이터
* 딥러닝에 적용하려면 연속적인 값으로 바꿔야함
* 임배딩으로 그 역할을 하더라도 연속적이지 않았던 것을 연속적으로 바꾸는 것이므로 여전히 제약이 존재
* 한국어 자연어 처리가 더욱 어려운 이유
* 1. 교착어
* 어근에 접사가 붙어 의미와 문법적 기능이 부여됨 -> 다양하게 파생가능
* 2. 띄어쓰기
* 추가적인 분절을 통해 띄어쓰기를 정제하는 과정이 필요함
* 3. 평서문과 의문문
* 의문문과 평서문이 같은 형태의 문장구조를 가지는 경우가 있음
* 4. 주어 생략
* 사람과 달리 컴퓨터는 문맥 정보로 생략된 정보를 메꾸기가 힘듦 -> 문장 이해에 어려움
* 5. 한자 기반의 언어

표어문자인 한자를 표음 문자인 한글로 표현하며 정보의 손실이 발생

2020/8/29

* 김기현의 자연어 처리 딥러닝 캠프 69p 까지 3시간 학습하였습니다.
* 아나콘다와 토치를 깔아 실습을 진행하였습니다.
* 자연어 처리는 초기에 n-gram 방식을 사용했음
* 2010년 RNN 방식으로 기존의 문제를 해결한 언어 모델을 고안함
* 나중에는 n-gram과 결합하여 더 나은 성능의 언어 모델을 만듦
* 그러나 기존에 언어 모델이 주로 쓰이던 음성 인식과 기계번역 분야에 적용하기에는 구조적 한계와 높은 연산량으로 성과를 거둘 수는 없었음
* word2vec가 발표되고 단순한 구조의 신경망을 사용하여 단어들을 잠재공간에 투사시키게 됨
* 기존에는 문장이란 단어들과 시퀀셜 데이터이므로 순환 신경망을 통해 해결해야 한다는 고정관념이 있었음
* CNN만을 활용한 텍스트 분류 모델을 임베딩 벡터와 결합하면서 성능이 극대화됨
* 2014년 seq2seq가 발표되고 어텐션 기법이 개발됨
* -> 주어진 정보에 기반하여 자유롭게 문장을 생성하는 자연어 생성이 가능해짐
* 뉴럴 튜링 머신 ( 특정 주소 뿐만 아니라 여러 주소에서 연속적으로 정보를 읽고 쓰는 방법) 이 제시되고,
* 이어 디퍼런셜 뉴럴 컴퓨터가 제시되었음 이러한 신경망을 통해 메모리를 활용하는 기법인 메모리 증강 신명망이 등장하여 QA와 같은 문제에 효율적으로 대응할 수 있을것으로 보임
* 변분 오토인코더, 생성적 적대 신경망을 통해 영상 처리 분야는 기존 판별 모델에서 벗어난 모델 학습으로 옮겨갔음
* 딥러닝에서 사용하는 손실 함수는 실제 기계번역을 위한 목적 함수와는 괴리가 있음
* 자연어 생성 분야에서는 강화학습을 활용하여 SeqGAN과 같이 GAN을 구현하는 방법이 제시되기도 함
* 강화학습의 폴리시 그레디언트 방식을 자연어 생성에 적용하여 영상 처리 분야의 적대적 학습과 같은 방법을 자연어 처리에서 흉내낼 수 있게 됨
* 강화학습을 통해 자연어 생성에서 목적 함수로부터 보상을 받을 수 있게 되어 문장 생성 능력이 더 극대화 됨
* 랜덤 변수 : 확률을 이야기할 때 런덤하게 발생하는 어떤 사건
* 랜덤 변수는 불연속적인 이산값인 경우가 많음
* 불연속적인 랜덤 변수를 다루는 이산 확률 분포는 확률 변수가 정의된 공간을 나타냄
* 불연속적인 이산 확률 변수에 대한 확률 함수를 확률 질량 함수라고 함
* 이산적인 확률 변수를 갖는 확률 분포로는 베르누이 분포와 멀티눌리 분포가 있음
* 베르누이 분포는 0과 1 두개의 값만 가질 수 있음 - 이항분포
* 멀티눌리 확률 분포는 여러 개의 이산적인 값을 가질 수 있음 - 다항분포
* 연속적인 값을 다루는 연속 확률 변수를 가지는 확률 분포는 연속 확률 분포라고 함
* 연속 확률 분포에서는 확률 밀도 함수를 통해 확률 밀도를 정의할 수 있음
* 결합확률은 두 개 이상의 사건이 동시에 일어날 확률을 의미 - 두 개 이상의 변수를 가짐
* 두 사건이 서로에게 영향을 끼치지 않을 경우 독립이라고 부름
* 조건부 확률은 하나의 확률 변수가 주어졌을 때 다른 확률 변수에 대한 확률 분포임
* 주변 확률 분포는 두 개 이상의 확률 변수의 결합 확률 분포가 있을 때, 하나의 확률 변수에 대해서 적분을 수행한 결과를 말함
* 기댓값은 보상과 보상을 받을 확률을 곱한 값의 총합 - p(x) 분포에 따른 가중평균
* 몬테카를로 샘플링은 랜덤 성질을 이용하여 임의의 함수 적분을 근사하는 방법임
* 몬티홀 문제에 대해서도 배웠음

2020/8/30

* 김기현의 자연어 처리 딥러닝 캠프 84p 까지 3시간 학습하였습니다.
* 수식위주로 공부하였습니다.
* 일반화 : 데이터로부터 결과값을 예측하는 것 (미지의 데이터에 대해 좋은 예측을 하는 것)
* -> 실제 확률 분포로부터 데이터를 수집하고 수집된 데이터를 가장 잘 설명하는 확률 분포 모델을 추정함으로써 알고자 하는 실제 확률 분포를 근사함
* 가능도 함수에서 가능도를 최대화 하도록 하는 파라미터를 추정하는 것을 최대가능도 추정(MLE)라고 함
* 소수점이 너무 작게 표현되면 언더플로 현상이 나타남
* 덧셈 연산은 곱셈 연산보다 빠름 -> 가우시안 분포에서 지수를 제거할 수 있음
* 가능도에 로그를 취하여 로그 가능도를 최대화 할 수 있음 -> 식에 -1을 곱하면 최소화 문제로 치환 가능
* - 음의 가능도라고 함
* 신경망 또한 확률 분포 함수임
* 신경망 가중치 파라미터가 훈련 데이터를 잘 설명하도록 하기 위해 경사 하강법을 사용할 수 있음
* 정보 이론 : 데이터를 정량화하기 위한 응용 수학 분야
* 정보량 : 불확실성 또는 놀람의 정도 - 발생 확률이 낮을수록 불확실성이 높고 발생 시 놀람이 커짐 -> 정보량이 높으면 발생 확률이 낮음
* 확률이 낮은 사건에 대해 서술한 문장이 맞을수록 그 정보는 소중해짐
* 엔트로피 : 정보량의 평균(기댓값)
* 엔트로피는 퍼져있는지 뾰족한지를 가늠할수 있는 척도 -> 엔트로피가 작으면 분포는 뾰족함
* 교차 엔트로피 : 분포 함수 P에서 샘플링한 x를 통해 분포 함수 Q의 평균 정보량을 나타낸 것
* P를 사용하여 대상 분포 Q의 엔트로피를 측정함
* 분포 P와 분포 Q가 비슷한 모양일수록 교차 엔트로피 H(P,Q)는 더 작은 값을 가짐
* -> 손실함수를 사용, 손실 함수의 값이 최소가 되도록 경사하강법을 통해 신경망을 훈련
* 교차 엔트로피가 최소가 되도록 경사하강법을 수정
* 보통의 연속 확률 분포는 평균제곱오차 손실 함수를 통해 훈련함
* 샘플에 대한 확률 값을 구하기 어려워 교차 엔트로피는 적용하기 힘듦 - 이산 확률 분포에서 교차 엔트로피를 수행
* 쿨백-라이블러 발산(KLD) : 두 분포 사이의 괴리를 보여줌 (P,Q)
* P와 Q의 위치에 따라 KLD값은 달라짐 - 대칭의 개념이 아니기에 거리라고는 표현하지 않음
* -> 두 분포 사이의 차이를 줄이고자 할 때 KLD를 최소화 하는 전략을 사용
* 교차 엔트로피를 최소화하는 것이 로그 가능도를 최대화 또는 음의 로그 가능도를 최소화하는 것과 동일함

MSE를 사용하는 것은 신경망이 반환하는 연속 확률 분포가 가우시안 분포를 따른다는 가정에 기반을 둠