10.

ANN은 딥러닝의 핵심이다. 다목적, 강력, 확장성이 뛰어나, 매우 복잡한 기계 학습에 이상적

이 장에서는 최초의 ANN 구조를 살펴보고 인공신경망을 소개한다.

MLP(Multi-Layer Perceptirons)을 제시하고, 텐서플로우를 사용해 MNIST숫자 분류를 해결

1. From Biological to Artificial Neurons

ANN이 우리 삶에 더 깊은 영향을 줄 것이라고 믿을 것들을 소개

* 현재 신경 네트워크를 학습하는 데 사용할 수 있는 방대한 양의 데이터가 있고, ANN은 크고 복잡한 문제에 대해 성능이 우수
* 컴퓨팅 능력의 증가로 합리적인 대규모 신경 네트워크 학습 가능
* 학습 알고리즘 개선
  1. Biological neurons

인공 뉴런 전에 생물학적 뉴런

* 뉴런은 시냅스를 통해 다른 뉴런의 신호라고 불리는 짧은 전기적 자극을 받는다. 뉴런이 수 밀리 초 이내에 다른 뉴런으로부터 충분한 수의 신호를 수신하면 자신의 신호를 작동
* 따라서 개개의 뉴런은 오히려 간단한 방식으로 행동하는 것처럼 보이지만, 실제로는 수십억 개의 뉴런으로 구성된 광대한 네트워크로 구성
* 복잡한 계산은 단순한 뉴런의 광대한 네트워크에 의해 수행 될 수 있다.
  1. Logical Computations with Neurons

Warren McCulloch와 Walter Pitts는 하나 이상의 이진(on/off)입력과 하나의 이진 출력을 가지는 인공 뉴런 모델을 제안

인공 뉴런은 특정 수 이상의 입력이 활성화 되었을 때, 출력을 활성화

이러한 단순화 된 모델을 사용해도, 원하는 모든 논리적 명제를 계산하는 인공 뉴런 네트워크를 구축하는 것이 가능하다는 것을 보여줌

* 1. The Perceptron

; 가장 단순한 ANN구조 중 하나 -> 선형 임계값 단위(LTU:Linear Threshold Unit)라고 부르는 약간 다른 인공 뉴런을 기반

Input&output 은 on/off대신 숫자가 되고, 각 입력의 연결은 가중치와 연결

LTU는 입력의 가중 합을 계산 -> 그 합에 step function을 적용-> 결과 출력

퍼셉트론에서 사용되는 가장 일반적인 step function은 Heaviside함수 (때때로 sign함수가 대신 사용)

LTU를 학습한다는 것은 그림 10-4에서 w0,w1,w2에 대한 올바른 값을 찾는 것을 의미

퍼셉트론은 LTU의 단일 층으로 구성, 각 입력에는 모든 뉴런이 연결 -> input neuron라고 불리는 pass through neurons이라고 표현됨

이는 모든 입력값을 출력

또한 추가 bias feature는 일반적으로 추가됨

이 bias feature는 일반적으로 항상 1을 출력하는 bias nouron이라는 특별한 유형의 뉴런을 사용해 표현

그림 10-5는 두개의 입력(x1,x2)과 세개의 출력을 갖는 퍼셉트론을 나타냄

이 퍼셉트론은 인스턴스를 동시에 세 가지 다른 이진 클래스로 분류 할 수 있으므로 다중 출력 분류기

그렇다면 어떻게 학습할 수 있을까?

동일한 출력을 가질 때마다 두 뉴런 사이의 연결 가중치가 증가

구체적으로, 퍼셉트론은 한 번에 하나의 학습 인스턴스로 공급되며, 각 인스턴스에 대해 예측

잘못된 예측을 산출한 모든 출력 뉴런에 대해 올바른 예측에 기여한 입력으로부터 연결 가중치를 강화

각 출력 뉴런의 결정경계(decision boundary)는 선형이므로 퍼셉트론은 복잡한 패턴을 학습 할 수 없지만, 학습 인스턴스가 선형으로 분리될 수 있다면 이 알고리즘이 솔루션으로 수렴된다는 것을 입증 -> Perceptron convergence theorem

Scikit-learn은 단일 LTU 네트워크를 구현하는 퍼셉트론 클래스를 제공 (iris dataset이용)

Scikit-learn의 퍼셉트론 클래스는 SGD Classifier와 같은 하이퍼 파라미터를 사용

Logistic Regression 분류기와 달리, 퍼셉트론은 클래스 확률을 출력하지 않는다!

오히려 퍼셉트론은 hard threshole값을 기반으로 예측

Perceptrons이라는 제목의 논문에서 퍼셉트론의 사소한 XOR분류와 같은 문제를 해결 할 수 없다는 사실을 강조 (그림 10-6의 왼쪽)

그러나 여러가지 퍼셉트론을 stacking하면, 퍼셉트론의 한계점 중 일부를 제거할 수 있다.

* 이러한 결과를 MLP(Multi-Layer Perceptron)이라고 부름

특히 MLP의 출력을 계산하여 확인할 수 있기 때문에, XOR문제를 해결 가능(그림 10-6 오른쪽)

* 1. Multi-Layer Perceptron and Backpropagation

MLP는 하나의 input layer와 hidden layer라 불리는 하나 이상의 LTU layer와, output layer라 불리는 LTU의 최종 layer로 구성

Output layer를 제외한 모든 layer는 bias neuron을 포함하며 다음 layer에 완전히 연결

ANN에 두 개 이상의 Gidden layer가 있을 경우 -> DNN(Deep Neural Network

각각의 학습 인스턴스에 대해 역 전파 알고리즘은 먼저 예측(forward pass)을 진행하고, 오류를 측정한 다음, 역으로 각 layer를 거쳐 각 연결에서 error 기여도를 측정(reverse pass)하고, 마지막으로 error를 줄이기 위해 연결 가중치를 미세하게 조절(Gradient Descent step)

저자는 이 알고리즘을 제대로 작동시키기 위해, MLP의 구조를 바꿈

Step function을 logistic function으로 대체 (step function은 평평한 세그먼트로만 구성되어 있기 때문에, gradient descent는 평평한 표면에서 움직일 수 없어 대체가 필수적임)

Logistic function외 다른 activation function을 사용 할 수 있다.

1. \_The hyperbolic tangent function tanh\_

S자형, 연속적, 미분 가능하지만 출력 값의 범위는 -1~1

각 layer의 출력이 훈련 시작시, more or less하게 정규화 경향

수렴 속도 높이는 데 종종 도움이 됨

1. \_The ReLU function

연속적이지만 z=0일 때 미분 불가능 (기울기가 갑자기 변하기 때문에, gradient descent가 바운스 될 수 있음)

하지만 실제로 매우 잘 작동, 계산 속도 빠름

다음 코드는 activation function와 도함수(derivative)를 보여준다.

1. Training an MLP with Tensorflow’s High-Level API

텐서플로우를 사용해 MLP를 학습하는 가장 간단한 방법은, scikit-learn-compatibla API를 제공하는 TF.Learn을 사용!

DNNClassifier클래스를 사용하면, hidden layer를 여러 개 사용하여 심층 신경 네트워크(DNN)를 쉽게 학습할 수 있고, 예상되는 클래스 확률을 출력 할 softmax output layer를 만들 수 있다.

아래 코드는 두개의 hidden layer와 10개의 뉴런을 가지고 있는 softmax output layer로 분류되도록 DNN을 학습

이 코드는 먼저 학습세트에서 실제 가치가 있는 열 집합을 만듦 -> DNNClassifier를 만듦 -> Scikit-Learn compatibility helper로 묶음 -> 마지막으로 50개의 인스턴스의 batch를 이용해, 40000번의 반복 학습

이 코드는 MNIST dataset에서 실행하면 실제로 98.2%의 정확도를 달성하는 모델 얻을 수 있음

1. Training a CNN Using Plain Tensorflow

네트워크 구조를 보다 잘 제어하려면, 텐서플로우의 lower-level python api를 사용하는 것이 좋다!

이번 섹션에서는 이 API를 사용하기 전과 같은 모델을 만들고 MNIST dataset에서 미니배치 gradient Descent를 구현

1단계 – 텐서플로우 그래프를 빌드하는 construction phase

2단계 – execution phase ->여기서 실제로 그래프를 실행하여 모델을 학습

* 1. Construction Phase

1단계부터 시작!

텐서플로우 라이브러리를 import

그런 다음 input & output 수를 지정

각 계층에서 hidden neurons의 수를 설정

다음은 그래프 reset을 정의했다.

Placeholder 노드를 사용하여 학습데이터와 타겟을 나타낼 수 있다

X의 shape는 부분적으로만 정의됨

첫 번째 차원의 인스턴스와, 두 번째 차원의 feature를 갖는 2차원 텐서(행렬)라는 것을 알 수 있다.

또한 feature의 개수는 28\*28라는 것을 알 수 있다.

But 각 학습 batch 에 몇 개의 인스턴스가 포함되는지 알 수 없다. 그렇기 때문에 X의 shape는 (none, n\_inputs)이 된다.

마찬가지로 y는 인스턴스 당 하나의 엔트리를 갖는 1차원 텐서가 될 것이지만, 이 시점에서 batch의 크기를 알 수 없기 때문에 shape는 None이다.

이제 실제 신경망을 만들자!

Placeholder X는 input layer로 작동 – 이는 execution phase에서 한 번에 하나의 학습 batch로 대체

이제 두개의 hidden layer와 output layer를 만들어야 함 – 두 개의 hidden layer는 거의 동일 차이점은 단지 연결되는 input과 그들을 포함하는 뉴런의 개수 정도

Ouput layer도 매우 유사하지만 ReLU활성화 함수 대신 softmax활성화 함수를 사용

위 코드에 대해 line별로 살펴보자

Neuron layer를 만들기 위한 함수를 생성했다!(neuron\_layer)

이 함수는 deep neural network를 만들기 위해 사용됨

첫 번째 hidden layer는 X를 입력으로 사용

두 번째 hidden layer는 첫 번째 hidden layer의 출력을 입력으로 사용

마지막으로, output layer는 두 번째 hidden layer의 출력을 입력으로 사용

(X x-> first hidden layer -> second hidden layer -> output layer)

위 코드는 명확성을 위해 name scope를 사용함

Softmax 활성화 함수를 거치기 전에 neural, network의 ouput

최적화를 위해 나중에 softmax계산 처리 할 것임

예상할 수 있듯이, 텐서플로우에는 standard neural network layers를 만드는 데 유용한 많은 기능을 제공,

아래 코드는 kernel과 bias로 명명되는 weight와 bias변수를 만들고 activation인수를 사용해 활성화 함수를 설정

Neuron\_layer() 대신 dense()함수를 사용

이제 신경망 모델을 사용할 준비가 되었으므로, 이를 학습시키는 데 사용할 비용함수를 정의

교차 엔트로피 사용! -> 목표 클래스에 대한 낮은 확률을 추정하는 모델에 패널티를 줌

텐서플로우에는 교차 엔트로피를 계싼하는 함수 제공

Sparse\_softmax\_cross\_entropy\_with\_logits()은 logits 을 기준으로 교차 엔트로피를 계산0

그리고 0에서 클래스의 수에서 1을 뺀 범위의 정수 형태인 label을 예상( 이경우 0-9)

이것은 각 인스턴스에 대해 교차 엔트로피를 포함하는 1차원 텐서를 제공

그 후 reduce\_mean()함수를 이용해, 모든 인스턴스에 대한 평균 교차 엔트로피를 계산

신경망 모델과 비용함수 가지고 있다!

이제 비용함수를 최소화하기 위해 모델의 파라미터를 조정하기 위한 GradientDescentOptimizer를 정의

Construction phase의 마지막으로 중요한 단계는 모델을 평가하는 방법을 지정 (우리는 단순히 accuracy이용)

먼저 각 인스턴스에 대해 가장 높은 logit이 target 클래스에 해당되는지에 대한 여부를 확인해 신경망의 예측이 올바른지 확인 -> in\_top\_k()함수 사용 ;1차원의 텐서를 반환

따라서 이 boolean값을 float형으로 캐스팅 한 다음 평균을 계산 -> 네트워크의 전체 accuracy를 얻을 수 있다.

그런 다음, 모든 변수를 초기화 -> 학습 된 모델 파라미터 변수를 디스크에 저장하는 saver 만듦

1. Input과 target에 대한 placeholder 만듦
2. Inuron layer를 만드는 함수 만듦
3. DNN을 만들기 위해 cost function 정의
4. 최적화 도구를 만듦
5. 성능 측정(performance measure)정의
   1. Execution phase

텐서플로우는 데이터를 가져와서(fetch) 0~1사이로 크기를 조정(scale)하고, 셔플링하며, 하나의 미니배치를 로드하는 간단한 함수를 제공하는 자체 helper를 제공

이제 실행하려는 epoch의 수와, mini-batch의 크기를 정의

위의 코드는 텐서플로우 세션을 열고 모든 변수를 초기화하는 init 노드를 실행

그런 다음 기본 학습 루프를 실행 -> 각 epoch에서 학습 세트 크기에 해당하는 mini-batch의 수만큼 반복

각 mini-batch는 next\_batch()메소드를 통해 데이터를 가져온 후,(fetch)학습 작업을 실행하고 현재 mini-batch의 input 데이터와 target을 feeding

그런 다음, 각 epoch의 마지막에서 last mini-batch와 전체 데스트 세트의 모델을 평가, 결과 출력

마지막으로 모델 파라미터 변수가 디스크에 저장

* 1. Using the Neural Network

신경망이 학습되었으므로, 이를 이용하여 예측 할 수 있다!

그러기 위해, 동일한 construction phase를 재사용할 수 있지만, 아래 코드와 같이 execution phase를 변경해보자

두 번째 줄의 코드는 디스크에서 모델 파라미터 변수를 로드

그런 다음 분류를 위한 새로운 이미지를 로드

이때 주의할 점은, 학습 데이터와 동일한 feature scaling을 해야함

그런 다음 logits 노드를 평가.

1. Fine-Tuning Neural network Hyperparameters

신경망의 유연성은 주요 단점 중 하나 – 이를 조정하기 위한 파라미터들이 있음

간단한 MLP에서도 layer의 수, layer당 뉴련의 수, 각 layer에서 사용할 활성화 함수 유형, 가중치 초기화 로직 등 변경할 수 있다.

교차 검증과 함께 그리드 서치를 사용해 올바른 파라미터 찾을 수 있음 -> but, 튜닝 할 매개변수가 askg고, 대규모 데이터 셋에서 신경망을 학습하는데 시간 많이 걸림

따라서 무작위 검색(randomized search)를 사용하는 것이 좋음!

다른 옵션으로는 Oscar과 같은 도구 사용

우선 hidden layer의 수부터 시작해보자!

* 1. Number of Hidden Layers

계층적 아키텍쳐는 DNN이 보다 좋은 솔루션으로 수렴할 수 있을 뿐 아니라 새로운 데이터 셋에서 일반화 할 수 있는 능력을 향상시킴

많은 문제에 대해 하나 또는 두개의 hidden layer로 시작 할 수 있다

예를 들어, 수백 개의 뉴런이 있는 hidden layer 하나를 사용해 MNIST데이터 셋에서 97%이상의 정확도에 쉽게 도달 할 수 있다

대략 동일한 양의 뉴런을 가진 두 개의 hidden layer를 사용해 대략 98%의 정확도를 얻을 수 있다.

좀 더 복잡한 문제의 경우, 학습 셋이 과적합(overfitting)이 시작될 때까지 hidden layer의 수를 점차적으로 늘릴 수 있다.

매우 복잡한 작업에는 일반적으로 수십 개의 layer가 있는 네트워크가 필요 또는 수백 개의 layer가 있지만, 완전히 연결되어 있지 않을 수도 있다.

But 중요한 것은 이러한 네트워크를 처음부터 학습할 필요 X

* 유사한 작업을 수행하는 미리 학습 된 최신의 네트워크의 일부를 재사용하는 것이 훨씬 더 일반적
* 이는 학습이 더 빨라지고, 적은 데이터를 필요로 함!
  1. Number of Neurons per Hidden Layer

Input/ouput layer의 뉴런 수는 작업에 필요한 input/output의 유형에 따라 결정됨

예를 들어 MNIST에는 28\*28=784의 input뉴런과, 10개(0~9)의 output뉴런을 필요로 함

Layer의 수와 마찬가지로 네트워크가 과적합이 시작될 때까지 점차적으로 뉴런의 수를 늘릴 수 있음

일반적으로 layer당 뉴런의 수보다 layer의 수를 증가시키는 것이 가치가 있다!

* 1. Activation Functions

대부분, hidden layer에 Relu 활성화 함수를 사용할 수 있다.

Output layer의 경우, softmax 활성화 함수는 일반적으로 분류 작업에 탁월