
솔라리스의 인공지능 연구실

1. 오토인코더(Sparse Autoencoder) 1 – AutoEncoders & Sparsity

2015년 12월 23일 by Solaris

오늘 살펴볼 내용은 Sparse Autoencoder 강의의 Autoencoders & Sparsity 챕터이다.

1.4 Autoencoders and Sparsity(오토 인코더와 희박성)

지금까지 알아본 Neural Network는 샘플 데이터의 입력값(x)과 목표값(y)이 모두 주어진 labeled data를 Training data로 사용했다. 이런 형태로 데이터의 입력값과 목표값이 모두 주어진 상태에서 학습하는 것을 감독학습(Supervised Learning)이라고 한다. 이와 반대로, 데이터의 입력값(x)만이 주어진 상태로 학습하는 방법-unlabeled data를 통한 학습- 도 존재한다.이런 형태로 데이터의 입력값만이 주어진 상태에서 데이터의 특징을 찾아내는 학습을 비감독학습(Unsupervised Learning)이라고 한다.

오늘 알아볼 **autoencoder**는 기존의 Neural Network의 Unsupervised learning 버전이다. autoencoder에서 출력값은 입력값의 개수와 같다.($y^{(i)} = x^{(i)}$)

autoencoder의 기본적인 구조는 아래와 같다.

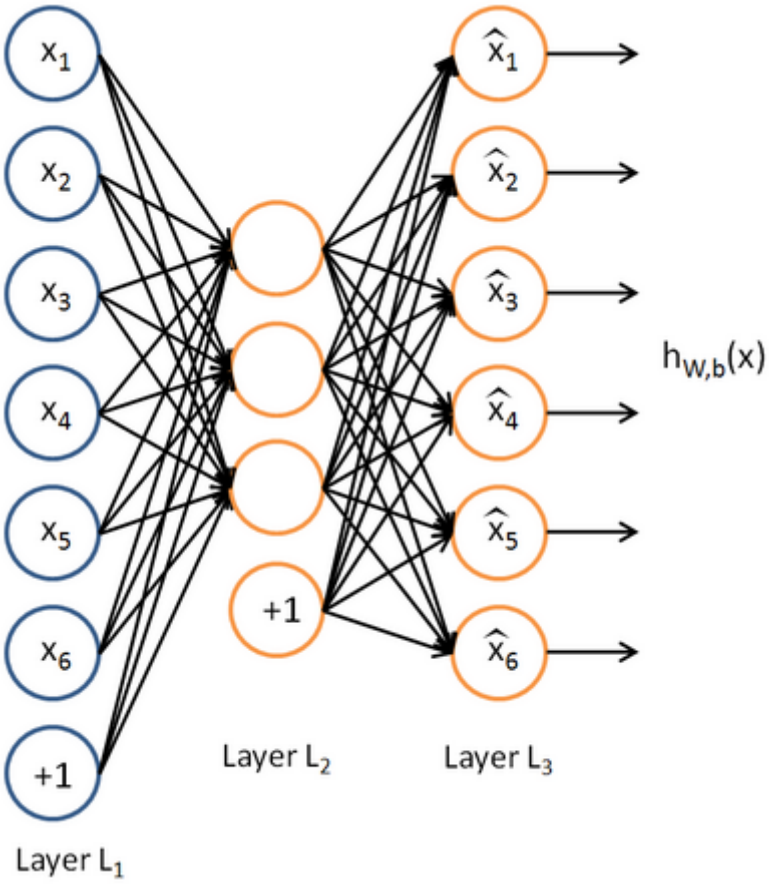

그림 1 – *a*utoencoder의 기본적인 구조

위와 같은 구조에서 autoencoder는 $h_{W,b}(x)$가 되는 함수를 학습하고자 한다. 다른 말로 하면, autoencoder는 항등함수(**역자 주**-모든 정의역 x에 대해서 $f(x) = x$를 만족하는 함수)의 근사를 학습하고자 한다. 즉, 출력값 $\hat{x}$를 입력값 x와 유사하게 만들고자 한다. 항등함수를 학습하는 것은 별로 어려운 일인것 처럼 보이지 않는다. 하지만 네트워크의 제한에 의해서 우리는 데이터의 흥미로운 구조(특징)를 발견할 수 있다.

구체적인 예를 들어 생각해보자. 만약 입력 데이터 값 x가 10×10 이미지(100 pixels)의 픽셀강도(pixel intensity)-픽셀의 밝기값-라면 $n = 100$이다. 이때 레이어 L_2의 히든 유닛의 개수 $s2$를 50이라고 하자. 출력값 y는 입력값과 같은 개수이므로 100이다.($y \in^{50}$)

이 때, 우리는 오직 50개의 히든 유닛을 가지고 있으므로 네트워크는 입력값의 *압축된* 표현을 학습해야만 한다.(**역자 주**-만약 히든 유닛이 100개라면 각각의 히든 유닛1개가 각각의 입력값에 대한 항등함수의 형태를 취하면 되므로 압축될 필요가 없다.-) 즉, 히든 유닛의 활성함수의 결과값(activations)이 $a^{(2)} \in^{50}$이므로, 히든 유닛의 활성화값(activations)은 입력값 x와 같이 100-pixel(100개)로 **복원(Reconstructed)** 되어야만 한다. 만약 입력값이 완벽하게 무작위적—각각의 x_i가 다른 특징들에 대해 IID Gaussian independent하게 추출되었다면— 이라면 이런 압축 절차는 매우 어렵다. 하지만 데이터에 일정한 구조가 있다면, 예를 들어서 입력값들의 특징간에 연관성이 있다면, autoencoder는 이런 연관성들을 찾아 낼 수 있을 것이다. 사실, 이런 저차원의 표현을 학습하는 간단한 autoencoder는 PCA들과 매우 유사하다.

위에서는 히든 유닛(s_2)의 개수가 작은 경우를 예로 들었지만, 히든 유닛의 개수가 많은 경우라도(심지어 입력 유닛의 개수보다 많은 경우에도), 우리는 네트워크에 제한을 걸음으로써 데이터의 흥미로운 구조(특징)를 발결한 수 있다. 특별히 우리는 히든 유닛에 **sparsity(희박)** 제한을 건다. 그러면, 히든 유닛의 개수가 많더라도 autoencoder는 데이터의 특징(흥미로운 구조)을 발견할 수 있을 것이다.

약식으로, 만약 뉴런(활성함수)의 결과 값이 1에 가까우면 뉴런이 active한 상태, 0에 가까우면 inactive한 상태라고 하자. 우리는 대부분의 상황에서 뉴런이 inactive하게 제한하고자 한다. 이런 표기는 활성함수가 시그모이드(sigmoid) 함수일 경우를 가정한 것이고 만약 활성함수로 tanh를 사용한다면, 활성함수의 결과값이 -1에 가까우면 inactive하다고 정의한다.

$a_j^{(2)}$가 autoencoder의 j번째 히든유닛의 activation(활성함수의 결과값)을 나타냈던 것을 기억하자. 하지만, 이 표기는 입력데이터 x와 활성함수의 결과값(activation)간의 상관성을 뚜렷하게 보여주지 않으므로 입력값 x에 대한 j번째 히든 유닛의 결과값을 $a_j^{(2)}(x)$로 다시 표기하자. 더욱이,

$$\hat{\rho_j} = \frac{1}{m}\sum_{i=1}^{m}[a_j^{(2)}(x^{(i)})] \tag{1}$$

*그림 **2** – 히든 유닛 **j**의 평균 활성화값**(activation)***

을 j번째 히든 유닛 의 평균 활성화값(activation)이라고 정의하자. (**역자 주**–ρ는 그리스 문자로 '로'라고 읽는다.) 우리는 대략적으로 아래와 같은 제한을 네트워크에 걸고자 한다.

$$\hat{\rho}_j = \rho \tag{2}$$

*그림 **3** – 네트워크 제한*

이때 ρ를 **sparsity parameter**라고 한다. 보통 ρ는 0에 가까운 작은 값을 부여한다. (우리는 $\rho = 0.05$로 지정하자.) 다른 말로 하면, 우리는 히든 유닛 j의 평균 활성화값(activation)을 0.05에 가깝게 만들고 싶은 것이다. 이 제약 조건을 만족시키기 위해서, 대부분의 히든 유닛의 활성함수의 결과값(activation)은 0 근처의 값이 되어야만한다

이 조건을 만족 시키기 위해서, 우리는 또 다른 제약조건을 걸 것이다. 우리의 최적화 방법론의 목적은 $\hat{\rho}_j$가 ρ로부터 상당히 빗나가면 벌점을 부여하는 것이다. 이런 제약 조건을 거는 방법은 여러가지 있다. 그 중에서 우리는 아래의 방법을 사용할 것이다.

$$\sum_{j=1}^{s_2} \rho log \frac{\rho}{\hat{\rho}_j} + (1-\rho) log \frac{1-\rho}{1-\hat{\rho}_j} \tag{3}$$

*그림 **4** – 최적화를 위한 제약조건*

여기서, s_2는 히든 레이어의 뉴런(노드)의 개수이고, j는 히든 유닛의 번호이다.

만약, 당신이 KL Divergence의 개념에 익숙하다면, 이 제약조건은 아래와 같은 형태로 쓰여질 수도 있다.

$$\sum_{j=1}^{s_2} KL(\rho || \hat{\rho}_j) \tag{4}$$

*그림 **5 – KL Divergence** 표기*

여기서 $KL(\rho||\hat{\rho}_j) = \sum_{j=1}^{s2} \rho log\frac{\rho}{\hat{\rho}_j} + (1-\rho)log\frac{1-\rho}{1-\hat{\rho}_j}$이다. 이것은 평균 ρ를 갖는 베르누이 확률변수(Bernoulli Random Variable)와 평균 $\hat{\rho}_j$을 갖는 베르누이 확률변수 간의 Kullback-Leibler(KL) Divergence이다. KL-divergence는 두개의 다른 분포가 얼마나 다른가를 측정하는 표준함수이다.(만약 당신이 전에 KL-Divergence에 대해서 들어본적이 없다면, 그것에 대해 걱정하지 말라. 당신이 알아야할 모든 것은 이 강의에 포함되어 있다.)

이 제약조건 함수는 $\hat{\rho}_j = \rho$일때 $KL(\rho||\hat{\rho}_j) = 0$, 그렇지 않은 경우에는 $\hat{\rho}_j$가 ρ로부터 벗어날수록 단조 증가하는 특징을 갖는다. 예를 들어, 아래의 그림에서, 우리는 $\rho = 0.2$로 설정하였다. 그리고 $\hat{\rho}_j$의 값의 변화에 따른 $KL(\rho||\hat{\rho}_j)$의 값의 변화를 그래프로 표현하였다.

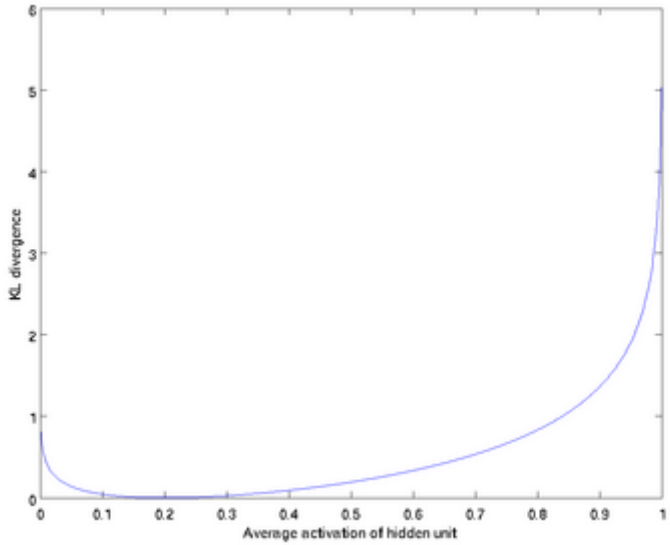

*그림 **6 – KL-divergence** 함수의 결과($\rho = 0.2$인 경우)*

우리는 $\hat{\rho}_j = \rho$ 일때, KL-divergence 함수의 결과 값이 최소값인 0에 도달하는 것을 볼 수 있다. 그리고 $\hat{\rho}_j$의 값이 0 또는 1에 가까워 질수록 KL-divergence의 값이 증가하는 것을 볼 수있다. 그러므로, 이 제약 조건 함수를 최소화하기 위해서 $\hat{\rho}_j$는 ρ에 근접해야만 한다. (**역자 주-**KL-divergence 함수를 통해 우리의 목적인 $\hat{\rho}_j = \rho$ 을 달성하기 위한 최적화 함수를 정의할 수 있다.)

이제 우리의 전체 비용함수를 정의하자. 전체 비용함수는 아래와 같다.

$$J_{sparse}(W,b) = J(W,b) + \beta \sum_{j=1}^{s_2} KL(\rho||\hat{\rho}_j) \tag{5}$$

*그림 **7 – Backpropagation**을 위한 비용함수 정의*

$J(W,b)$는 전 강의에서 정의되었다.(**역자 주-**tutorial의 이전 챕터를 참조하라.) β는 sparsity 제한 함수의 비율을 조정한다. $\hat{\rho}_j$는 함축적으로 W에 의해 결정된다. 또한 b는 히든 유닛 j의 평균 activation이다. 그리고 히든 유닛의 activation은 파마리터 W,b에 의존한다.

KL-divergence 제한조건을 전체 비용함수의 미분식으로 통합하기 위해서, 코드의 약간의 수정을 통한 간단한 구현 트릭을 사용한다. 특별히, 두번째 레이어(l=2) 이전에, backpropagation 동안에 다음을 계산할 수 있다.

$$\delta_i^{(2)} = (\sum_{j=1}^{s_2} W_{ji}^{(2)} \delta_j^{(3)}) f'(z_i^{(2)}) \tag{6}$$

*그림 **8 –** 레이어 **2**의 **Backpropagation** 과정*

이제 위 식 대신에 아래 식을 사용한다.

$$\delta_i^{(2)} = ((\sum_{j=1}^{s_2} W_{ji}^{(2)} \delta_j^{(3)}) + \beta(-\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i})) f'(z_i^{(2)}) \tag{7}$$

*그림 **9 –** 수정된 레이어 **2**의 **Backpropagation** 과정*

이 조건을 계산하기 위해 알아야 하는 것은 $\hat{\rho}_i$이다. 그러므로, 학습데이터의 평균 activation을 계산하기 위해서 어떤 예제(example)의backpropagation을 계산하기 전에 전방향(forward pass) 계산을 먼저해야 한다. 만약 학습 데이터 세트가 컴퓨터 메모리에 부담이 되지 않을 정도로 충분히 작다면(이것은 이후 제시될 프로그래밍 과제의 상황이다.), 당신은 모든 예제에 대해서 전방향 계산을 하고 activation들을 메모리에 저장 한채로 $\hat{\rho}_i$들을 계산할 수 있다. 그러면 당신은 backpropagation을 수행하기 위해서 미리 계산된 activation들을 사용할 수 있다. 만약 데이터 세트가 메모리에 저장하기 너무 크다면, 당신은 예제들(examples)을 훑으면서 각각의 전방향 계산을 하면서 각각의 activation들을 더하고 그 뒤 $\hat{\rho}_i$를 계산해야 한다. (전방향 계산의 결과값은 $\hat{\rho}_i$을 계산하기 위한 activation들-$a_i^{(2)}$-을 구한 다음에 버린다.) 그리고 계산된 $\hat{\rho}_i$값을 구한뒤, 각각의 예제들에 대해서 전방향 계산을 다시 한다. 그러면 그 예제에 대한 backpropagation을 얻을 수 있다. 이런 후자의 경우에는, 각각의 예제들에 대해 전방향 계산을 두번 해야만 한다. 따라서, 이것은 계산에 있어서 비효율성을 발생시킨다.

위의 알고리즘의 gradient descent method(경사하강법)에 의한 전체 유도과정은 이번 강의의 범위를 뛰어넘는다. 하지만 만약 당신이 이 방식으로 수정된 backpropagation 방법을 이용해서 autoencdoer를 구현한다면, 당신은 우리의 목적인 $J_{sparse}{}^{(W,b)}$에 대한 gradient descent를 수행하게 된다. 당신은 derivative checking method을 통해서 이것을 스스로 확인 할 수도 있다.

정리

이번 강의에서는 Neural Network의 Unsupervised 버전이라고 할 수 있는 autoencoder의 기본 개념과 autoencoder의 학습을 위해 필요한 Sparsity 개념에 대해서 알아보았다.

간략하게 요약하자면, autoencoder는 Neural Network의 unsupervised 버전이다. 따라서 Output Layer의 노드의 개수가 Input Layer의 노드의 개수와 같다.(Unsupervised learning에서는 출력값의 개수가 입력값의 개수와 같아야 한다.)

autoencoder를 사용하는 목적은 데이터의 특징을 찾아내는 것이다. 따라서, autoencoder가 의미 있는 결과를 내려면 Neural Network와 마찬가지로 학습이 잘 되어야만 한다. 따라서, 학습을 위한 방법론으로 Sparsity 개념이 제안되었다. Sparsity는 희박이라는 문자에서 알 수 있듯이 히든 레이어의 활성함수의 결과값을 대부분 inactive(0을 출력)하게 만드는 조건이다.

Sparsity 개념을 이용해서 Neural Network의 경우와 마찬가지로 Backpropagation 방법을 이용해서 autoencoder를 학습 시킬 수 있다.

원문

http://ufldl.stanford.edu/wiki/index.php/Autoencoders_and_Sparsity

*본 포스팅은 **Stanford** 대학의 **Andrew Ng** 교수가 제공하는 **UFLDL Tutorial** 강의 자료를 부분적으로 한글 번역하고 주석을 덧붙여 완성한 자료임을 알려드립니다.*

Posted in Deep Learning, UFLDL

‹ Game AI #2 – Reinforcement Learning – Introduction

안드로이드: 기기별 해상도와 UI 최적화 ›

5 THOUGHTS ON "1. 오토인코더(SPARSE AUTOENCODER) 1 – AUTOENCODERS & SPARSITY"

핑백: 1. 오토인코더(Sparse Autoencoder) 1 – AutoEncoders & Sparsity – 솔라리스의 인공지능 연구실 – 인공지능 블로그

YB

2017년 3월 5일 12:57 오후

안녕하세요 요즘 Deep learning을 공부하는 학생입니다.
CNN말고 autoencoder에도 관심이 가서 배우고 있는 중인데, 혹시 autoencoder는 단독으로 쓰이면 어디에 사용되고,
CNN같은 다른 신경망 구조랑 겸해서 사용도 가능한가요?? 마냥 차원축소라는 개념만 들어서는 그냥 compression한다는 생각밖에 안들어서요 ㅠ

응답

Solaris

2017년 3월 5일 1:31 오후

안녕하세요.

Autoencoder 단독으로 사용할 경우 말씀하신대로 Compression 과정을 통한 Feature Extraction으로 사용하는 경우가 많고요.

이렇게 추출한 유의미한 Feature를 인풋으로 SVM이나 Neural Networks등에 집어 넣어서 CNN처럼 Classification 등에 사용할 수 있습니다.

CNN과 같이 사용할 경우 위에 언급했듯이 CNN Input에 Autoencoder로 구한 Feature Vector를 추가하는 형태로 사용할 수 있습니다.

이 글은 딥러닝이 부상하는 초기시기에 학습 교재로 집필된 자료인데요.

최근에는 Variational Autoencoder 구조를 이용해서 Generative Model로 사용하는게 트렌드입니다.

응답

YB

2017년 3월 6일 2:16 오후

아 답변 감사합니다:) Autoencoder라는 개념을 배우고 있는데 설명이 도움이 됐습니다 ㅎㅎ 감사합니다!

응답

익명

2017년 3월 5일 10:26 오후

Autoencoder는 1) Data Compression, 2) Data Visualization 그리고 3) Deep Neural Network를 Pre-Train 하는 방법으로 사용됩니다.

응답

댓글 남기기

이메일은 공개되지 않습니다.

댓글

이름

이메일

웹사이트

댓글 달기

검색 …

최근 글

NGUI 버튼 위로 파티클(Particle) 뿌리기

6. 텐서플로우(TensorFlow)를 이용해 간단한 DQN(Deep-Q-Networks) 에이전트를 만들어 보기 (CatchGame)

5-1. 텐서플로우(TensorFlow)를 이용해 자연어를 처리하기(NLP) – Word Embedding(word2vec)

1-1. 윈도우(Windows) 환경에서 docker를 이용해서 Tensorflow설치하기

4. 텐서플로우(TensorFlow)를 이용한 ImageNet 이미지 인식(추론) 프로그램 만들기

최근 댓글

Solaris (1-1. 윈도우(Windows) 환경에서 docker를 이용해서 Tensorflow설치하기)

형태희 (Game AI #1 – Pathfinding – A* Algorithm을 중심으로)

Q (1-1. 윈도우(Windows) 환경에서 docker를 이용해서 Tensorflow설치하기)

YB (1. 오토인코더(Sparse Autoencoder) 1 – AutoEncoders & Sparsity)

익명 (1. 오토인코더(Sparse Autoencoder) 1 – AutoEncoders & Sparsity)

글 목록

2017년 2월

2016년 8월

2016년 5월

2016년 1월

2015년 12월

2015년 11월

2015년 10월

카테고리

Android

Deep Learning