Fundamentals of Deep Learning

Chapter 2. 전방향 신경망 학습

2.1. 패스트 푸드 문제

우리는 딥러닝을 통해 여러 가지 흥미로운 문제를 어떻게 풀 수 있는지 이해하기 시작했지만 파라미터 벡터(신경망 내 모든 연결에 대한 가중치)가 무엇이 되어야 하는지 어떻게 알아낼 수 있느냐 하는 한 가지 큰 의문이 여전히 남아 있음. 이것은 일반적으로 학습이라는 과정을 통해 알아낼 수 있음. 학습하는 동안 신경망에 많은 학습 예제를 가르치고 그 학습 예제들에 대한 오차를 최소화하기 위해 반복적으로 가중치를 수정함. 충분한 예제를 학습했다면 신경망은 학습된 과제를 매우 효율적으로 해결할 수 있게 됨.

앞 장에 언급한 패스트푸드 예제 및 선형 뉴런과 관련된 예를 계속 생각해보자. 간단히 되짚어 보면, 패스트푸드 식당에서 햄버거, 감자튀김, 탄산음료로 구성된 세트 메뉴를 사고, 세트 메뉴의 단품 개수에 따라 돈을 냄. 가격이 얼마나 할지 알고 싶지만, 단품이 각각 얼마인지는 가격표에 나와 있지 않고 유일한 정보는 직원이 알려주는 세트의 가격임. 이 문제를 해결하기 위해 하나의 선형 뉴런을 학습시키고자 함.

한 가지 아이디어는 지능적으로 학습 데이터를 선택하는 것. 즉, 주문 하나에 음식 하나씩 주문하는 방식으로 첫 주문에 햄버거 하나만, 다음 번에 감자튀김 하나만, 마지막에는 탄산음료 한 잔 이렇게 말임. 지능적으로 학습 데이터를 선택하는 것은 일반적으로 아주 좋은 아이디어임. 학습 데이터를 영리하게 처리함으로써 신경망의 학습 효율을 더 높일 수 있다는 연구는 많음. 하지만 오로지 이 방법만 사용할 때 문제는 실제 상황에서 거의 결과를 얻을 수 없다는 점임.

대신, 일반적으로 잘 동작하는 해결 방법을 찾아보자. 아주 큰 집합의 학습 데이터가 있다고 하자. 그러면 신경망이 간단한 수식을 사용해 i번째 학습 데이터에서 출력을 계산할 수 있음. 가능한 최적의 가중치 즉, 학습 데이터에서 발생하는 오차를 최소화하는 가중치를 선택하기 위해 뉴런을 학습시키고자 함. 이때 만나게 되는 모든 학습 데이터의 오차 제곱을 최소화하기를 원한다고 하자. 좀 더 수학적으로, t가 i번째 학습 데이터에 대한 올바른 답이고 y는 신경망에 의해 계산된 값이라면 오차 함수 E의 값을 최소하히길 원함.

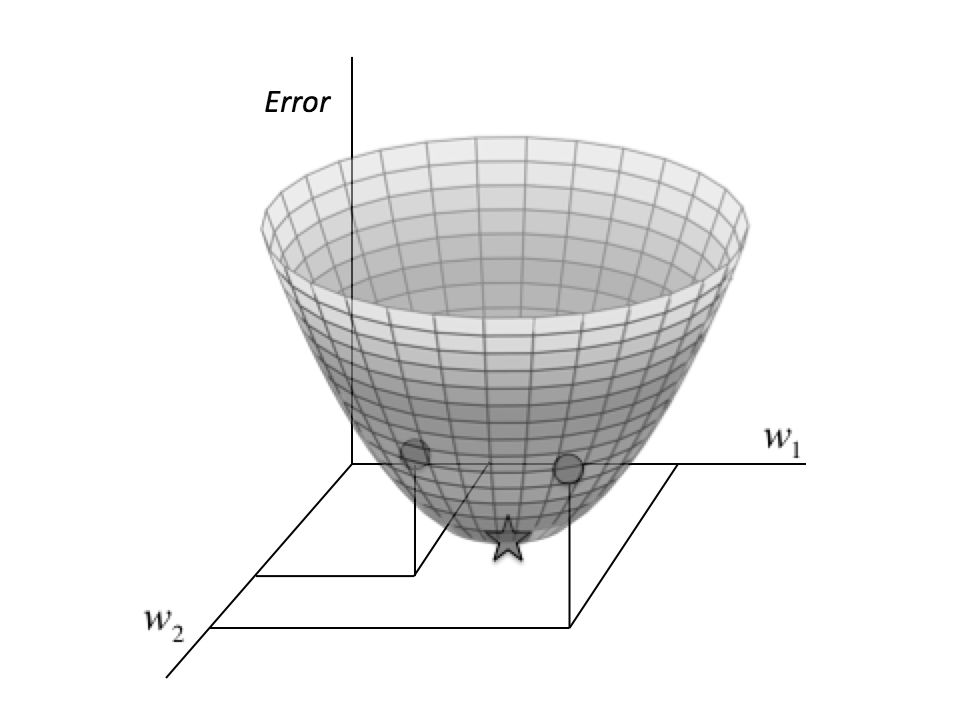
모델이 모든 학습 데이터에 대해 완벽하고도 정확하게 예측할 때 오차의 제곱은 0임. 또, E가 0에 가까우면 가까울수록 좋은 모델임. 결론적으로, 우리의 목표는 E가 가능한 한 0에 근접하게 하는 파라미터 벡터 를 선택하는 것임.

이 시점에서 연립 방정식으로 이 문제를 다룰 수 있는데, 왜 굳이 오차 함수로 우리 스스로를 괴롭혀야 하는지 궁금할 것임. 결국, 우리는 한 뭉치의 미지수들을 가지고 있고 여러 방정식의 집합을 가짐. 일관성 있는 학습 데이터 집합을 가졌다고 가정하면 오차는 자동으로 0이 됨.

이 통찰은 현명한 생각이지만, 불행하게도 일반화하기 어려움. 비록 여기서 선형 뉴런을 사용하고 있지만, 선형 뉴런은 학습시키는 데 제약이 있어서 실제로는 잘 사용되지 않는다는 점을 기억하기 바람. 또한 앞장 끝부분에서 소개한 시그모이드와 tanh, ReLu 같은 비선형 뉴런을 사용하기 시작하는 순간 더는 연립 방정식을 만들 수 없음.

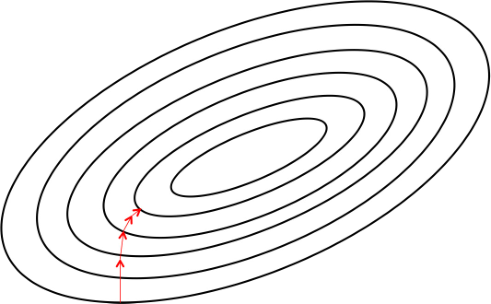
2.2. 경사 하강법

문제를 단소화해 모든 학습 예제의 오차 제곱을 최소화할 방법을 시각화해보자. 선형 뉴런이 두 개의 입력만을 가진다고 하자. 그러면 가중치 w1과 w2로 이루어진 수평 차원과 오차 함수 E로 만들어진 수직 차원의 3차원 공간을 상상할 수 있음. 이 공간에서 수평 평면의 점들은 가중치들의 다른 설정에 해당하고, 이 점들에서 높이는 발생한 오차에 해당함. 모든 가능한 가중치를 만드는 오차를 고려하면 밑의 그림과 같이 3차원 공간에서 사발 모양의 곡선을 얻게 됨.



또한, 이 곡면을 타원 등고선들의 집합으로 편리하게 시각화할 수 있는데, 여기서 최소 오차는 타원의 중심에 있음. 이 설정으로 우리는 두 가중치로 표현되는 2차원 평면에서 문제를 다룸. 등고선들은 같은 E 값을 가지는 w1과 w2의 조합에 대응됨. 등고선이 서로 가까울수록 경사가 가파름. 실제로 가장 가파른 하강 방향은 항상 등고선에 대해 수직임. 이 방향은 경사(gradient)라는 벡터로 표현됨.

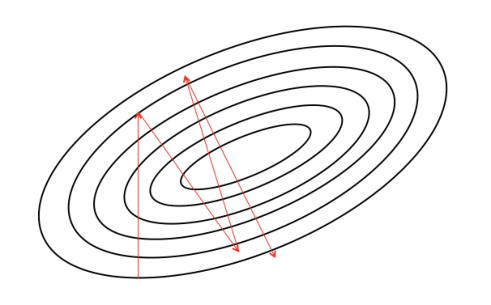
이제 오차 함수의 값이 최소인 가중치 값을 찾는 방법에 대한 수준 높은 전략을 개발할 수 있음. 신경망의 가중치들을 무작위로 초기화하고 수평면 어딘가에 있다고 가정해보자. 현재 위치에서 경사를 구해 가장 가파른 경사 방향을 찾을 수 있으며 해당 방향으로 한 걸음 나갈 수 있음. 그러면 원래 있던 곳보다 최소값에 더 가까운 새로운 위치에 있음을 알게 됨. 이 새로운 위치에서 가장 가파른 경사의 방향을 다시 구할 수 있고 이 새로운 방향으로 또 한 걸음 나아감. 이 과정을 밑의 그림과 같이 간단하게 시각화할 수 있다. 이 저냑을 따르면 결과적으로 최소 오차 지점에 도달하게 됨. 이 알고리즘을 경사 하강법(gradient descent)이라고 하는데, 우리는 이 결사 하강법을 사용해 개별 뉴런을 학습시키는 문제와 전체 신경망을 학습시키는 좀 더 일반적인 과제를 풀어봄.



2.3. 델타 규칙과 학습률

패스트푸드 뉴런을 학습시키는 정확한 알고리즘을 도출하기 전에 하이퍼파라미터(hyperparameter)에 대해 간단히 언급하겠음. 신경망에서 정의한 가중치 파라미터 외에도 학습 과정을 수행하기 위해서는 학습 알고리즘 또한 추가로 한 쌍의 파라미터가 필요한데, 이를 하이퍼파라미터라 하고 그중 하나가 바로 학습률임.

실제로는 등고선에 수직으로 이동하는 각 단계에서 새로운 방향을 재계산하기 전에 얼마나 멀리 나갈지를 결정해야 함. 이 거리는 곡면의 가파른 정도에 의존하는데 왜 그럴까? 최소값에 가까울수록 앞으로 나가는 거리는 더 짧아짐. 곡면이 많이 평평해지면 최소값에 가까워졌다는 것을 알 수 있음. 그래서 최소값에 얼마나 가까운지에 대한 지표로써 가파른 정도를 사용할 수 있음. 하지만 오차 곡면이 매우 완만하다면 학습하는 데 시간이 오래 걸릴 가능성이 있음. 하지만 오차 곡면이 매우 완만하다면 학습하는 데 시간이 오래 걸릴 가능성이 있음. 그래서 종종 경사에 학습률 e를 곱함. 밑의 그림에서 보듯이 학습률을 선택하는 것은 어려운 문제임. 앞에서 살펴본 것처럼 학습률이 너무 낮으면 아주 긴 학습 과정을 감수해야 함. 그래서 학습률이 너무 높으면 최소값 근처에서 발산할 가능성이 높음. 다음 장에서는 학습률을 선택하는 과정을 자동화하기 위해 학습률 적응을 활용한 다양한 최적화 기법을 배움.



자, 드디어 선형 뉴런 학습에 대한 델타 규칙(delta rule)을 도출할 준비가 됨. 각 가중치를 어떻게 바꿀지 계산하기 위해 경사를 평가함. 경사는 각 가중치에 대한 오차 함수의 편도함수(partial derivative)로, 다음과 같은 수식으로 표현할 수 있음.

가중치를 변경하는 이 방식을 모든 반복에 적용하면 드디어 경사 하강법을 사용할 수 있음.

2.4. 시그모이드 뉴런의 경사 하강법

이 절과 다음 절에서는 비선형을 활용한 신경망과 뉴런 학습을 다룸. 모델로는 시그모이드 뉴런을 사용하고 다른 비선형 뉴런에 대한 수식 유도는 독자들을 위한 연습 과제로 남김. 이 경우 바이어스 항목을 사용해도 되지만, 단순화하기 위해 뉴런들이 바이어스 항목을 사용하지 않는다고 가정함. 바이어스는 단지 값이 항상 1인 입력 연결의 가중치라고 가정하면 됨.

다음처럼 입력으로부터 출력 값을 계산하는 로지스틱 뉴런에 의한 동작 방식을 상기해보자.

뉴런은 입력의 가중된 합인 로짓 z를 계산함. 그 다음 최종 출력인 y를 계산하기 위해 입력 함수에 해당 로짓을 제공한다. 다행히도 이 함수들은 학습이 쉽도록 도와주는 아주 좋은 도함수를 가지고 있음. 학습을 위해 가중치들에 대한 오차 함수의 경사를 계산하려면 입력과 가중치들에 대한 로짓 함수의 도함수를 다음과 같인 취함.

또한, 로짓에 대한 출력의 도합수를 출력 항목으로 표현한다면 놀랍게도 다음과 같이 매우 간단함.

그리고 각 가중치에 대한 출력의 도함수를 얻기 위해 연쇄 법칙을 다음과 같이 사용함.

이 모든 것을 종합해 각 가중치에 대한 오차 함수의 도함수를 계산할 수 있음.

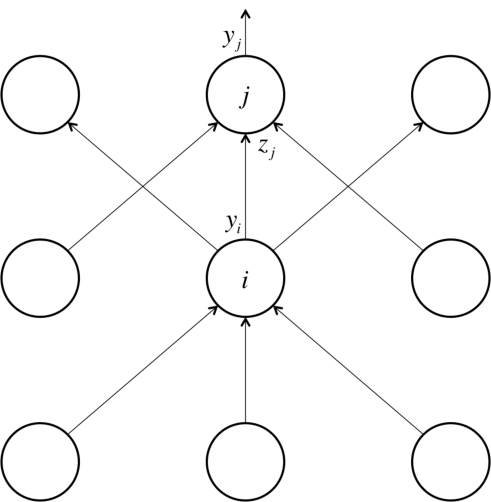
따라서 가중치를 수정하기 위한 최종 규칙은 다음과 같음.

여기서 보듯이 새로 수정된 규칙은 시그모이드 뉴런의 로지스틱 요소를 설명하기 위해 추가로 곱해진 항목을 제외하고 델타 규칙과 똑같음.

2.5. 역전파 알고리즘

자, 단순한 한 층의 뉴런들 대신 다층 신경망을 학습시키는 문제를 해결할 준비가 됨. 이를 해내기 위해 역전파(backpropagation)라는 방식을 사용하겠음. 역전파 알고리즘에는 어떤 개념이 숨겨져 있을까? 역전파 알고리즘에서 은닉 유닛(hidden unit)이 무엇을 해야 하는지는 알지 못하지만, 은닉층의 활성도를 바꿀 때 얼마나 빨리 오차가 변하는지는 계산할 수 있음. 이것으로 개별 연결의 가중치를 바꿀 때 오차가 얼마나 빨리 변하는지를 알 수 있음. 궁극적으로 가장 가파른 하강 방향을 찾으려 노력하는데, 이때 문제점은 매우 고차원의 공간에서 문제를 다루게 된다는 것임. 하나의 학습 예제에 대한 오차 도함수(error derivative)를 계산하는 것으로 시작해보자.

각 은닉 유닛은 많은 출력 유닛에 영향을 미칠 수 있음. 따라서 유용한 방법으로 오차에 대한 여러 개별 효과를 결합해야만 함. 우리의 전략은 일종의 동적 프로그래밍임. 은닉 유닛의 한 개 층에 대한 오차 도함수를 얻고, 이를 아래층의 활성도에 대한 오차 도함수를 계산하는 데 사용함. 그리고 은닉 유닛의 활성도에 대한 오차 도함수를 찾으면 은닉 유닛들로 이어지는 가중치에 대한 오차 도함수를 구하기가 매우 쉬움. 논의하기 쉽게 몇 가지 표기법을 새롭게 정의해보겠음. 밑의 그림을 살펴보자.



위 그림에서 아래 첨자는 뉴런의 층을 나타냄. 기호 y는 늘 그렇듯이 한 뉴런의 활성도를 나타냄. 비슷하게 기호 z는 뉴런의 로짓을 나타냄. 동적 프로그래밍 문제의 기본 사례를 살펴보는 것으로 시작하자. 특히 출력층에서 오차 도함수를 다음과 같이 계산함.

이제 귀납적 단계를 다루어 보자. j번째 층에 대한 오차 도함수가 있다고 가정해보면, 목표는 그 아래층인 i번째 층에 대한 오차 도함수 계산임. 이를 위해 i번째 층의 뉴런 출력이 j번째 층의 모든 뉴런의 로짓에 얼마나 영향을 미치는지에 대한 정보를 모아야 함. 이것은 아래층에서 들어오는 출력 데이터에 대한 로짓의 편미분이 단지 로 표현되는 연결 가중치라는 사실을 이용해 다음과 같이 표현할 수 있음.

또한, 다음과 같이 표현할 수 있음.

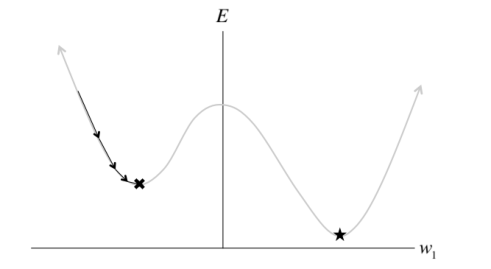
이 두 식을 결합하면 결과적으로 j번째 층의 오차 도함수 관점에서 i번째 층의 오차 도함수를 다음과 같이 표현할 수 있음.

그 다음 모든 편도함수로 적절히 테이블을 채우는 전체적인 동적 프로그래밍 과정을 통해 가중치에 대해 오차가 얼마나 변화하는지를 결정할 수 있음. 이것은 각 예제를 학습한 후 가중치가 어떻게 수정되는지를 다음과 같이 알려줌.

결국, 이 알고리즘을 완성하기 위해 이전과 마찬가지로 데이터셋에 전체 학습 예제에 대한 편도함수를 합산함. 이것은 다음과 같이 수정된 식으로 나타낼 수 있음.

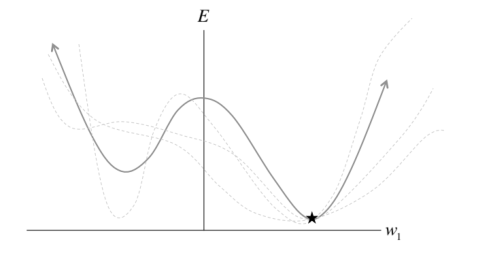
2.6. 확률적 경사 하강법과 미니배치 경사 하강법

2.5에서 알고리즘을 설명할 때 배치 경사 하강법이라는 경사 하강법의 한 버전을 사용했음. 배치 경사 하강법의 기본 개념은 전체 데이터셋을 사용해 오차 곡면을 계산한 후 경사를 따라 가장 가파른 하강 경로를 취하는 것임. 단순한 이차 오차 곡면일 때는 이것이 잘 작동하지만, 대부분 오차 곡면은 훨씬 더 복잡할 수 있음. 다음 그림을 살펴보자.



위 그림의 시나리오는 단일 가중치만 있고, 최적 설정을 찾기 위해 임의로 초기화하고 배치 경사 하강법을 사용함. 하지만 오차 곡면은 평평한 구간을 가지고 있으며, 운이 없으면 경사 하강법을 수행하는 동안 스스로 갇힐 수 있음.

또 다른 가능한 방식으로 확률적 경사 하강법(stochastic gradient descent, SGD)이 있음. 확률적 경사 하강법은 각 반복에서 오차 곡면이 단일 예제에 대해서만 추정됨. 밑의 그림에서 이 방식을 보여주는데, 여기에서는 단일 정적 오차 곡면 대신 동적 오차 곡면임. 결과적으로, 이 확률적 곡면상에서 하강하는 것은 평평한 구간을 탐색하는 능력을 크게 향상하게 함.

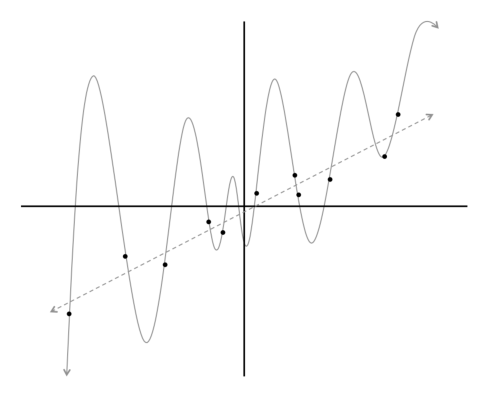


그러나 확률적 경사 하강법의 가장 큰 함정은 한 번에 한 가지 예에서 발생한 오차를 보는 것이 오차 곡면에 대한 충분히 좋은 근사가 아닐 수 있다는 것임. 이런 이유로 경사 하강법 계산에 상당한 시간이 걸리게 됨. 이 문제를 해결할 한 가지 방법은 미니배치 경사 하강법을 사용하는 것임. 미니배치 경사 하강법에서는 반복마다 전체 데이터셋의 부분 집합에 대해 오차 곡면을 계산함. 이 부분 집합을 미니배치(mini-batch)라고 하며, 미니배치의 크기는 학습률과 함께 또 다른 하이퍼파라미터가 됨. 미니배치는 배치 경사 하강법의 효율성과 확률적 경사 하강법에 의해 제공된 지역 최소값 회피 사이에서 균형을 유지함. 역전파와 관련한 가중치 갱신 단계는 다음과 같음.

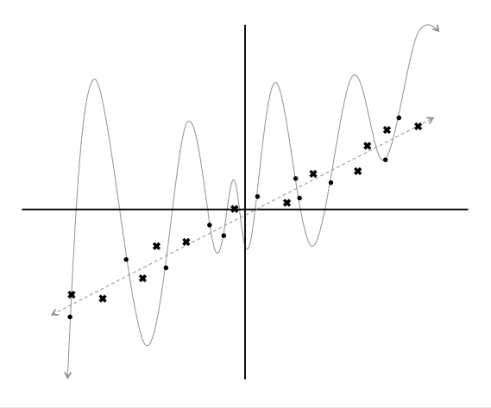
이것은 앞 절에서 유도한 식과 동일하지만, 데이터셋의 모든 예제를 합산하는 대신에 현재 미니배치 안의 예제들만 합산함.

2.7. 테스트 데이터와 검증 데이터 그리고 과적합

인공 신경망의 주요 이슈 중 하나는 모델들이 너무 복잡하다는 것임. 예를 들면, MNIST 데이터베이스에서 이미지 데이터를 가져와 30개의 뉴런으로 이루어진 두 은닉층에 공급하고, 마지막에는 10개 뉴런의 소프트맥스층에 도달하는 신경망을 생각해보자. 이 신경망에서 파라미터의 전체 수는 거의 25,000개임. 이것은 상당히 문제가 될 수 있음. 다음의 그림을 살펴보자.

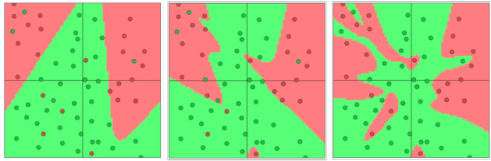


한 평면 위에 다수의 데이터 점을 주고, 이 데이터셋을 가장 잘 설명할 수 있는 곡선을 찾는 것이 우리의 목표임. 같은 데이터를 사용해 선형 모델과 12차다항식 모델을 학습시킴. 어떤 곡선이 더 믿음직스러운가? 대부분 점을 지나지 않는 선형 직선인가 아니면 데이터셋의 모든 점을 지나가는 복잡한 곡선인가? 이 시점에서는 훨씬 덜 인위적으로 보이는 직선을 신뢰해야 함. 하지만 정말 확실히 하기 위해 데이터셋에 더 많은 데이터를 추가해보자. 그 결과는 밑의 그림에 나와 있음.

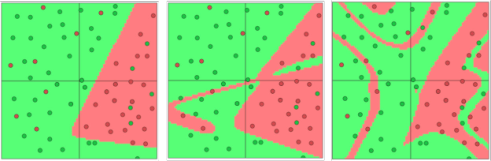


정성적으로만이 아니라 정량적으로 판단해도 선형 모델이 더 좋음. 그러나 이것은 머신러닝 모델을 평가하고 학습시키는 것에 관한 아주 흥미로운 지점으로 안내함. 아주 복잡한 모델을 만들면 학습 데이터셋에 완벽하게 맞추기가 매우 쉬움. 학습 데이터의 관찰 결과를 맞추기 위해 곡선 자체를 뒤트는 충분한 자유도를 모델에게 주었기 때문임. 그러나 새로운 데이터를 그런 복잡한 모델로 평가하면 매우 형편없이 동작함. 다시 말해, 이 모델은 일반화를 제대로 못 한 것임. 이런 현상을 과적합(overfiting)이라고 하며, 머신러닝 엔지니어라면 반드시 해결해야 하는 가장 큰 문제 중 하나임. 많은 뉴런을 포함한 수많은 층을 가진 신경망인 딥러닝에서 이것은 아주 심각한 문제가 됨. 이 모델들에서 연결은 그 수가 수백만 게에 달하며 가히 천문학적임. 그 결과 과적합은 아주 흔하게 일어남.

신경망의 맥락에서 이것이 어떻게 보이는지 살펴보자. 두 개의 입력과 두 가지 크기의 소프트맥스 출력, 3, 6, 20개의 뉴런을 가진 은닉층으로 구성된 신경망이 있다고 하자. 밑의 그림은 이 신경망을 미니배치 경사 하강법을 사용해 학습시키고 ConvNetJS를 이용해 시각화한 결과를 나타냄.



이 이미지들은 망의 연결 수가 증가할수록 데이터에 지나치게 과적합 하는 경향이 있다는 것을 아주 명확히 보여줌. 신경망을 깊게 만들 때도 유사한 과적합 현상을 볼 수 있는데, 밑의 그림에서 그 결과를 볼 수 있음. 여기서는 각각 3개의 뉴런으로 된 은닉층이 1, 2, 4개씩 있는 신경망을 사용함.



여기서 크게 세 가지 관찰 결과를 얻을 수 있음. 첫째, 머신러닝 엔지니어는 과적합과 모델의 복잡성 사이의 직접적인 균(trade-off)을 항상 고려하며 일함. 모델이 충분히 복잡하지 않으면, 문제를 해결하는 데 필요한 유용한 정보를 모두 잡아낼 만큼 강력하지 않을 수 있음. 그러나 모델이 아주 복잡하다면 과적합의 위험을 감수해야 함. 딥러닝은 복잡한 모델로 아주 복잡한 문제를 풀고, 과적합 방지를 위해 추가 대책을 적용하는 접근 방법을 취함. 이 장뿐 아니라 다음에 이어질 장들에서도 이런 많은 조치를 볼 수 있음.

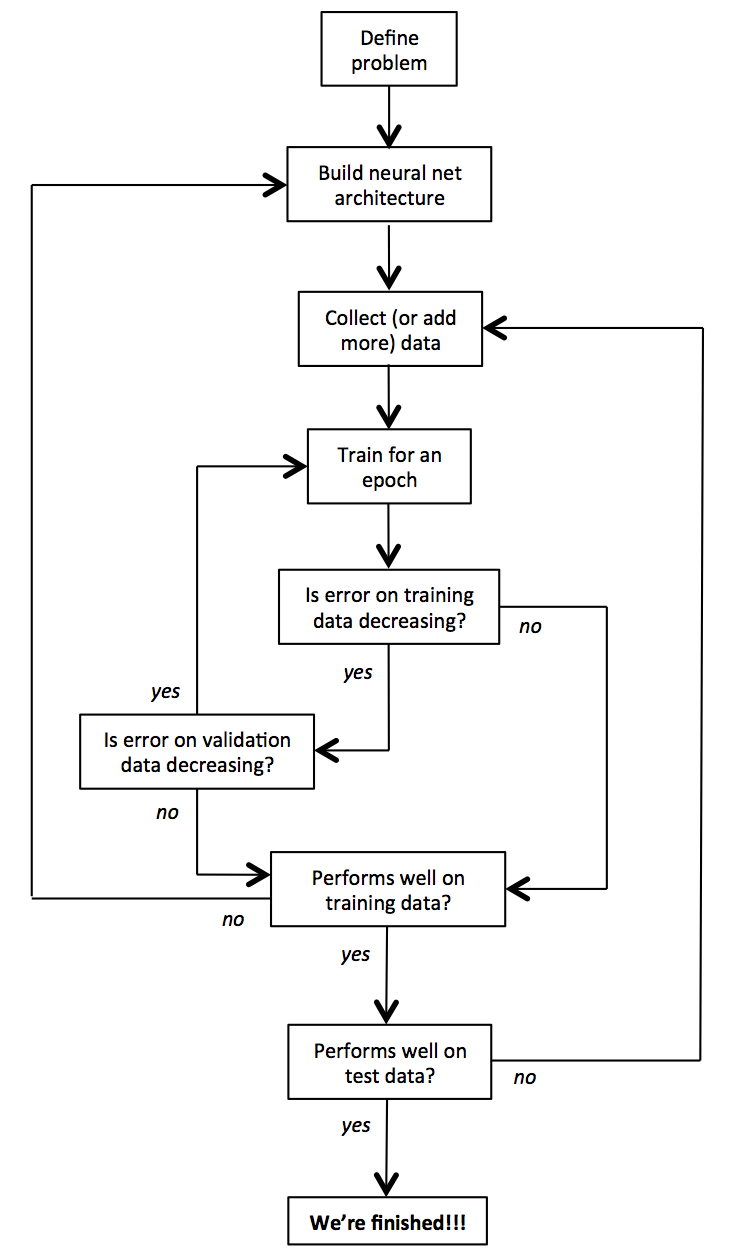
둘째, 학습에 사용한 데이터를 이용해 모델을 평가하는 것은 오류의 소지가 있음. 위의 선형 모델과 12차 다항식의 예제를 사용할 경우 12차다항식 모델이 선형 모델보다 적합하다는 거짓 결과를 얻게 됨. 결과적으로, 데이터셋 전체를 모델에 학습시키는 일은 거의 없음. 대신 학습 데이터와 테스트 데이터로 분리함.

학습 데이터와 테스트 데이터의 분리는 아직 보지 못한 데이터를 얼마나 잘 일반화하는지 직접 측정함으로써 모델의 공정한 평가를 가능하게 함. 실세계에서는 대규모 데이터셋을 구하기 힘들기 때문에 학습 과정에서 모든 데이터를 마음껏 사용하지 않는 것이 낭비처럼 보일 수 있음. 따라서 테스트 데이터로 학습 데이터를 재사용하거나 테스트 데이터를 만드는 절차를 무시하는 것에 매우 솔깃 할 수 있음. 하지만 테스트 데이터가 잘 구성되지 않으면, 모델로부터 의미 있는 어떠한 결론도 도출해 낼 수 없음.

셋째, 데이터를 학습시키는 동안 유용한 특징들을 학습하는 대신에 학습 데이터에 지나치게 맞춰져 과적합 되기 시작하는 시점이 있을 가능성이 높음. 이것을 피하려면 과적합이 시작되자마자 학습을 멈출 수 있으면 좋음. 이것은 형평없는 일반화를 방지하기 위함임. 이를 위해 학습 과정을 에포크(epoch)로 나눔. 한 에포크는 학습 데이터 전체에 대한 한 번의 반복 과정임. 다시 말해, 학습 데이터 크기는 d고, 미니배치 경사 하강법의 배치 크기를 b라고 하면, 한 에포크는 d/b번의 모델 갱신과 같음. 우리는 각 에포크의 끝에서 모델이 얼마나 잘 일반화됐는지를 측정하고 싶음. 이를 위해 추가로 검증 데이터를 사용함. 한 에포크의 끝에서 이 검증 데이터는 모델이 아직 알지 못하는 데이터를 사용할 때 어떻게 하는지를 말해 줄 것임. 검증 데이터를 사용할 때 정확도가 동일하게 유지되는 동안 학습 데이터를 사용할 때의 정확도가 계속 증가한다면 과적합이 이루어지고 있으므로 학습을 멈춰야 하는 시간임을 알려주는 좋은 신호임.

또한, 검증 데이터는 하이퍼파라미터 최적화 과정에서 정확도 대리 측정(proxy measure)에 도움이 됨. 앞에서 몇 개의 하이퍼파리미터에 대해 다루었지만, 아직 이 하이퍼 파리미터들의 최적값을 찾는 방법에 관한 틀을 전개하지는 않았음. 최적 설정값을 찾는 한 가지 방법은 격자 탐색(grid search)을 적용하는 것임. 이는 각 하이퍼파라미터에 대해 정해진 선택지의 집합에서 하나의 값을 고르고, 해당 하이퍼파라미터들의 모든 가능한 조합으로 모델을 학습시킴. 검증 데이터에 대해 최적 성능을 보이는 하이퍼파라미터 조합을 선택하고, 테스트 데이터에 대해 최적의 조합으로 학습된 모델의 정확도를 확인함.

과적합과 직접 맞서는 다양한 방법에 대해 살펴보기 전에, 앞의 내용을 바탕으로 딥러닝 모델을 만들고 학습시킬 때 우리가 사용할 작업 흐름에 대해 간략히 설명하겠음. 이 작업 흐름은 밑의 그림에 자세히 설명되어 있는데, 그리 복잡하지는 않지만 우리가 신경망을 제대로 학습시키는지 확인하려면 이 파이프라인을 이해하는 것이 대단히 중요함.



우선 문제를 엄밀히 정의하자. 이것은 입력과 잠재적인 출력, 이 둘의 벡터화된 표현을 정하는 것을 포함함. 예를 들어, 목표가 암을 찾는 딥러닝 모델을 학습시키는 것이라고 하자. 입력은 RGB 이미지가 되고, 이 이미지는 각 픽셀값을 하나의 벡터로 표현할 수 있음. 출력은 다음과 같이 상호 배타적인 3개의 확률에 대한 확률분포 값이 될 수 있음.

1. 정상
2. 양성 종양(아직 전이되지 않은 암)
3. 악성 종양(다른 장기로 이미 전이된 암)

문제를 정의한 후 이 문제를 풀기 위해 신경망 구조를 만들어야 함. 입력층은 이미지로부터 가공되지 않은 데이터를 받아들일 수 있는 적절한 크기여야 하며, 출력층은 크기 3인 소프트맥스여야 함. 또한, 망의 내부 구조(은닉층의 수, 연결 등)도 정의해야 함. 4장에서 합성곱(convolution) 신경망에 관해 이야기할 때 이미지 인식 모델의 구조에 대해 더 논의할 것임. 지금은 학습과 모델 만들기를 위해 상당한 양의 데이터를 수집해야 할 시점임. 이 데이터는 아마도 의학 전문가가 분류한 균일한 크기의 병리학적 이미지 형태일 것임. 이 데이터들을 학습 데이터와 검증 데이터, 테스트 데이터로 나누어 섞음.

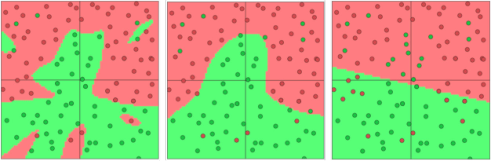
마지막으로, 경사 하강법을 시작할 준비가 되었음. 한 번에 한 에포크의 학습 데이터를 모델에 학습시킴. 각 에포크 처리가 끝나면 학습 데이터와 검증 데이터에 대한 오차가 감소하는 것을 확인함. 이 두 가지 오차 중 하나가 더 이상 개선되지 않을 때 테스트 데이터에 대한 모델의 성능이 충분한지를 확인하고 종료함. 만족스럽지 못하다면 구조를 다시 생각해보거나 우리가 원하는 예측을 하는 데 필요한 정보가 수집한 정보에 있는지를 다시 생각해 봐야 함. 학습 데이터에 대한 오차가 더는 개선되지 않는다면 데이터에 서 중요한 특징을 더 잘 파악해야 함. 검증 데이터에 대한 오차가 더 이상 개선되지 않는다면 과적합을 막는 조치들을 취해야 할 것임. 하지만 학습 데이터에 대한 모델의 성능이 만족스럽다면, 학습된 모델이 이전에 본 적 없는 테스트 데이터에 대한 성능을 측정할 수 있음. 테스트 데이터에 대한 모델의 성능이 만족스럽지 않다면, 테스트 데이터가 학습시킨 데이터에서 잘 표현되지 않은 예제 유형으로 구성된 것처럼 보이므로 데이터셋에 더 많은 데이터가 필요함.

2.8. 신경망에서 과적합 막기

학습 과정에서 생기는 과적합을 막기 위해 제안된 몇 가지 기술이 있는데, 이 절에서는 이 기술들에 대해 알아봄. 과적합 문제와 싸우는 한 가지 방법으로 ‘정형화(regularization)’라는 방법이 있음. 정형화는 큰 가중치 값들을 불리하게 하는 부가적인 항을 추가해 최소화하는 목적 함수를 수정함. 다시 말해, 목적 함수를 가 되게 바꿈. 여기서 가 커지면 가 커지며, 는 정형화의 강도를 나타냄. 선택한 값이 얼마나 과적합을 방어할지를 결정하는데, 값이 0일 때는 과적합 가능성에 대해 어떠한 조치도 취하지 않는다는 의미임. 값이 아주 크다면 모델이 학습 데이터에 대해 좋은 성능을 내는 파라미터 값을 찾는 것보다 를 가능한 작게 유지하는 것을 우선시한다는 것임. 결국, 값 선택은 매우 중요한 일이며 시행착오가 필요할 수 있음.

머신러닝에서 정형화의 가장 보편적인 형태는 L2 정형화임. 이것은 신경망에서 모든 가중치의 제곱 크기로 오차 함수를 증가함으로써 구현할 수 있음. 다시 말해, 신경망에서 모든 가중치 w에 대해 항을 오차 함수에 추가함. L2 정형화는 최고점인 가중치 벡터를 매우 불리하게 하고 분산된 가중치 벡터들을 선호하는 직관적 해석을 내포함. 이것은 신경망이 입력 중 일부를 많이 사용하는 것보다 오히려 조금씩 모든 입력을 사용하게 유도하는 매력적인 속성을 가지고 있음. 특히 L2 정형화를 사용하면 경사 하강법을 갱신하는 동안 결국 모든 가중치가 선형적으로 0으로 감소함을 의미하는 데 주의해야 함. 이 현상 때문에 L2 정형화는 일반적으로 가중치 감쇠라고도 함.

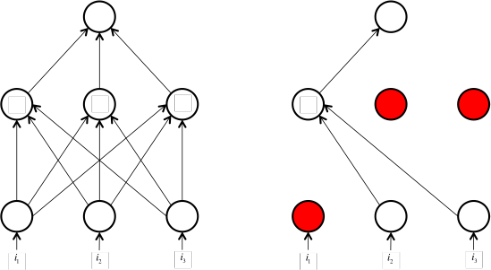
ConvNetJS로 L2 정형화의 효과를 시각화할 수 있음. 위에 기술했던 신경망을 표현하는 두 그림과 유사하게 2개의 입력과 크기가 2인 소프트맥스 출력, 20개의 뉴런을 포함하는 은닉층으로 구성된 신경망을 사용함. 미니배치 경사 하강법을 사용해 신경망을 학습시키고, 정형화 강도를 0.01, 0.1, 1로 함. 밑의 그림은 해당 결과를 보여줌.



정형화의 또 다른 보편적인 형태는 L1 정형화임. 여기서 신경망의 모든 가중치 w에 대해 를 추가함. L1 정형화는 최적화하는 동안 가중치 벡터를 드문드문하게 만드는 아주 흥미로운 특성이 있음. 다시 말해, L1 정형화로 뉴런들은 가장 중요한 입력의 작은 부분 집합만을 사용하게 되고 입력 노이즈를 매우 잘 견디게 됨. 비교해보면 L2 정형화로 만든 가중치 벡터는 일반적으로 분산된 작은 수임. L1 정형화는 어떤 특징들이 판정에 기여하고 있는지를 정확히 이해하고자 할 때 매우 유용함. 이런 특징 분석 수준이 필요 없다면 경험적으로 L2 정형화가 더 잘 수행되므로 L2 정형화를 사용하는 것이 더 좋음.

최대 노름 제약조건(max norm constraint)은 가 매우 커지는 것을 제한하는 것과 비슷한 목적이 있지만, 좀 더 직접적으로 이를 수행함. 최대 노름 제약조건은 모든 뉴런에 대한 입력 가중치 벡터의 크기에 절대 상한선을 적용하고, 제약조건을 강제 도입하기 위해 투영된 경사 하강법을 사용함. 다시 말해, 경사 하강법의 단계에서 입력 가중 벡터를 와 같이 움직일 때마다 해당 벡터는 반지름이 c인 구 위에 투영됨. 이때 c의 일반적인 값은 3과 4임. 이것의 좋은 점 중 하나는 가중치 갱신이 항상 제한되기 때문에 파라미터 벡터가 통제 불능 상태가 될 수 없다는 것임.

드롭아웃(dropout)은 과적합을 막기 위한 매우 다른 방법으로, 심층 신경망(deep neural network, DNN)에서 과적합을 막는 데 가장 선호되는 기법의 하나임. 학습 중에 드롭아웃은 어떤 확률 p(하이퍼파라미터)로 뉴런의 활성 상태를 유지하거나 0으로 설정해 구현함. 직관적으로 이것은 특정 정보가 없을 때도 망을 정확하게 만들고, 망이 뉴런의 어느 하나에 너무 의존적이 되는 것을 방지함. 좀 더 수학적으로 표현하면, 기하급수적으로 많은 다른 신경망 구조를 근사적으로 결합하는 방법을 제공해 과적합을 효과적으로 방지함. 이 드롭아웃 과정은 밑의 그림과 같이 표현됨.



드롭아웃은 꽤 직관적으로 이해할 수 있지만, 고려해야 할 몇 가지 중요하고 복잡한 사항이 있음. 먼저, 테스트 시 뉴런의 출력이 학습할 때 기대했던 출력과 일치하길 원하는데, 이 문제는 테스트할 때 출력 크기를 조정함으로써 해결할 수 있음. 예를 들어, p가 0.5라면 학습 과정에서 동일한 출력이 되도록 뉴런은 테스트할 때 출력을 반으로 나누어야만 함. 이것은 한 개의 뉴런 출력이 1 – p의 확률로 0으로 설정되기 때문에 알기 쉽다. 이것은 드롭아웃 전 한 뉴런의 출력이 x였다면 드롭아웃 후 예상되는 출력이

가 됨을 의미함. 하지만 테스트할 때 뉴런 출력의 크기 조정이 필요하므로 경험 없이 드롭아웃을 구현하는 것은 바람직하지 않음. 테스트에 걸리는 시간에 대한 성능은 모델 평가에서 대단히 중요함. 따라서 테스트 시간 대신 학습 시간에 크기 조정이 발생하는 역드롭아웃을 사용하는 것이 좋음. 역드롭아웃에서 비활성화되지 않은 뉴런은 값이 다음 층으로 전파되기 전에 p의 확률로 나눈 출력을 갖게 됨. 이런 수정을 통해

가 되고, 테스트할 때 뉴런의 출력 크기를 임의로 조정하는 것을 피할 수 있음.