**Chapter 10. Introduction to Artificial Neural Networks**

1. **From Biological to Artificial Neurons** 
   1. **Biological Neurons**
   2. **Logical Computations with Neurons**
   3. **The Perceptron**
   4. **Multi-Layer Perceptron and Backpropagation**
2. **Training an MLP with TensorFlow’s High-Level API**
3. **Training a DNN Using Plain TensorFlow** 
   1. **Construction Phase**
   2. **Execution Phase**
   3. **Using the Neural Network**
4. **Fine-Tuning Neural Network Hyperparameters** 
   1. **Number of Hidden Layers**
   2. **Number of Neurons per Hidden Layer**
   3. **Activation Functions**

**Chapter 10. Introduction to Artificial Neural Networks**

지적인 기계를 만드는 방법에 대한 영감을 얻기 위해 뇌의 구조를 바라보는 것은 인공 신경망 (ANN)에 영감을 주었던 핵심 아이디어입니다.

ANN은 점차적으로 그들의 생물학적 사촌 들과는 상당히 다른 것으로되었다. 일부 연구자들은 심지어 생물학적으로 유추 할 수있는 시스템으로 우리의 창의력을 제한하지 않도록 생물학적 유추를 모두 삭제해야한다고 주장합니다 (예 : "뉴런"보다는 "단위"라고 말함).

ANN은 Deep Learning의 핵심입니다.

다재다능하고 강력하며 확장 성이 뛰어나 수십억 개의 이미지(Google 이미지) 분류, 음성 인식 서비스(애플의 Siri) 강화, 최고의 비디오 추천 등 크고 매우 복잡한 기계 학습 작업에 이상적입니다.

이 장에서는 최초의 ANN 구조를 간략히 살펴보고 인공 신경망을 소개합니다. 그리고 MLP (Multi-Layer Perceptron)를 제시하고 TensorFlow를 사용하여 MNIST 숫자 분류 문제를 해결합니다(3장에서 scikit-learn으로 소개).

1. **From Biological to Artificial Neurons**

놀랍게도, ANN는 꽤 오랫동안 주변에 있었다.

ANN은 신경 생리학자인 워렌 맥컬록(Warren McCulloch)와 수학자 월터 피츠(Walter Pitts)에 의해 1943년에 처음 소개되었다. McCulloch와 Pitts는 획기적인 논리를 사용하여 복잡한 계산을 수행하기 위해 생물학적 뉴런이 동물의 두뇌에서 어떻게 작용할 수 있는지에 대한 간략한 계산 모델을 제시 하였고, 이것은 최초의 인공 신경 네트워크 구조였습니다. 그 이후로 많은 다른 구조가 발명되었습니다.

- 1960 년대까지 ANN의 초기 성공은 곧 우리가 진정으로 지적인 기계와 대화 할 것이라는 대폭적인 믿음을 이끌어 냈습니다. 이 약속이 실현되지 않을 것이라는 점이 명백 해지자 (적어도 상당 기간 동안) 자금이 다른 곳으로 날아 갔고 ANN은 길게 어두운 시대로 들어갔다.

- 1980 년대 초에는 새로운 네트워크 아키텍처가 발명되고 더 나은 교육 기술이 개발됨에 따라 ANN에 대한 관심이 되살아났습니다.

- 1990 년대에는 SVM과 같은 강력한 대체 기계 학습 기술이 더 나은 결과와보다 강력한 이론적 토대를 제공하는 것처럼 보였으므로 대부분의 연구자가 선호했습니다. 마지막으로 ANN에 대한 또 다른 관심이 있습니다

ANN이 우리 삶에 훨씬 더 깊은 영향을 줄 것이라고 믿을만한 몇 가지 이유가 있습니다.

- 이제는 신경망을 훈련시키는 데 사용할 수있는 방대한 양의 데이터가 있으며 ANN은 매우 크고 복잡한 문제에 대해 다른 ML 기술을 자주 능가합니다.

- 1990 년대 이후 컴퓨팅 파워가 엄청나게 증가한 덕분에 합리적인 시간에 대규모 신경 네트워크를 학습 할 수 있었습니다. 이것은 부분적으로 무어의 법칙에 의한 것이지만 수백만 명의 강력한 GPU 카드를 생산 한 게임 업계 덕분입니다.

- 훈련 알고리즘이 개선되었습니다. 공평하기 위해서 그들은 1990 년대에 사용 된 것들과 약간 다르지만, 이러한 비교적 작은 비틀기는 엄청난 긍정적 영향을 미칩니다.

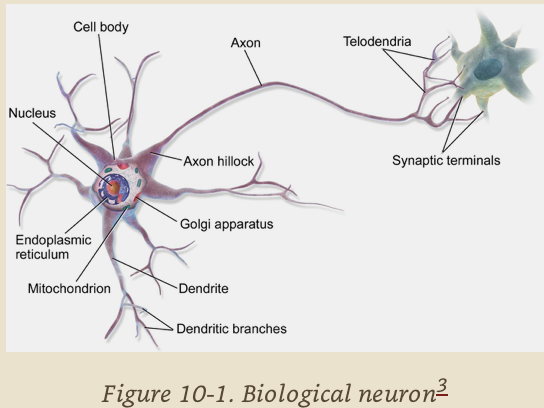
- ANN의 이론적 한계 중 일부는 실제로는 큰 문제가 아닌 것(양성)으로 밝혀졌습니다. 예를 들어, 많은 사람들은 ANN 교육 알고리즘이 지역 최적의 상태에 머물러있기 때문에 운명에 부딪혔다고 생각하지만 실제로는 거의 사용하지 않는것으로 나타났습니다 (또는 그러한 경우에는 대개 글로벌 최적에 근접).

- ANN은 자금조달과 진보의 선순환으로 들어간 것 같습니다. ANN을 기반으로하는 놀라운 제품은 정기적으로 헤드 라인 뉴스를 작성하여 점점 더 많은 관심과 기금을 모으고 있어 점점 더 발전하고 있습니다.

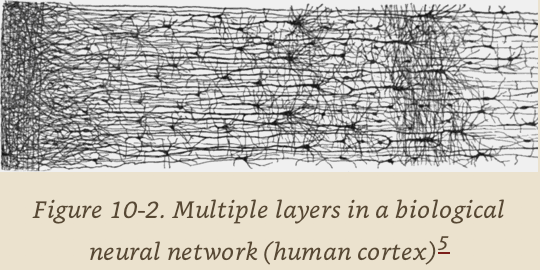
* 1. **Biological Neurons**

인공 뉴런에 대해 논의하기 전에 생물학적 뉴런 (그림 10-1)을 간단히 살펴 보겠습니다.

이것은 주로 동물의 대뇌 피질에서 발견되는 비정상적인 세포로, 핵과 세포의 복잡한 구성 요소의 대부분을 포함하는 세포체와 수상 돌기라고 불리는 많은 분지 확장 및 하나의 매우 긴 축색 돌기를 포함한다. 말단 근처에서 축색 돌기는 텔로덴드리아(telodendria)라고 불리는 많은 가지로 나뉘며, 이 지점의 끝에는 다른 뉴런의 수상 돌기(또는 세포체에 직접 연결되어있는) 인 시냅스 터미널(또는 시냅스)이라고 하는 소액 구조가 있습니다. 생물학적 뉴런은이 시냅스를 통해 다른 뉴런의 신호라고 불리는 짧은 전기 충격을 받습니다. 뉴런이 수 밀리 초 이내에 다른 뉴런으로부터 충분한 수의 신호를 수신하면 자신의 신호를 작동 시킵니다.

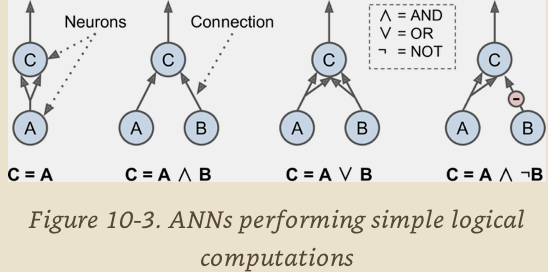


따라서 개개의 생물학적 뉴런은 오히려 간단한 방식으로 행동하는 것처럼 보이지만, 수십억 개의 뉴런에 일반적으로 연결된 각 뉴런의 광대한 네트워크에서 조직됩니다. 고도로 복잡한 계산은 매우 단순한 뉴런의 막대한 네트워크에 의해 수행 될 수 있습니다. 복잡한 개미가 단순한 개미의 결합 된 노력에서 나타날 수있는 것처럼. 생물학적 신경망 (BNN)의 구조는 여전히 활발한 연구 대상이지만 뇌의 일부분이 매핑되어 있으며 그림 10-2에서 볼 수 있듯이 뉴런은 종종 연속적인 계층으로 구성되는 것처럼 보입니다.



* 1. **Logical Computations with Neurons**

워렌 맥컬록 (Walren McCulloch)과 월터 피츠 (Walter Pitts)는 생물학적 뉴런의 매우 간단한 모델을 제안했는데 나중에 인공 뉴런으로 알려지게 되었다. 하나 이상의 이진 (on / off) 입력과 하나의 이진 출력을 가진다. 인공 뉴런은 특정 수 이상의 입력이 활성화되었을 때 출력을 활성화합니다. McCulloch와 Pitts는 그러한 단순화된 모델을 사용해도 원하는 모든 논리적 명제를 계산하는 인공 뉴런 네트워크를 구축 할 수 있음을 보여주었습니다. 예를 들어, 두 개 이상의 입력이 활성화되었을 때 뉴런이 활성화되었다고 가정하고 다양한 논리 연산을 수행하는 몇 개의 ANN을 작성해 봅시다 (그림 10-3 참조).



* 왼쪽의 첫 번째 네트워크는 단순히 identity funtion입니다.  
  뉴런 A가 활성화되면 뉴런 C도 활성화됩니다 (뉴런 A에서 두 개의 입력 신호를 받기 때문에). 그러나 뉴런 A가 꺼져 있으면 뉴런 C가 꺼집니다.
* 두 번째 네트워크는 논리 AND를 수행합니다.   
  뉴런 C는 뉴런 A와 B가 활성화된 경우에만 활성화됩니다 (단일 입력 신호로는 뉴런 C를 활성화하기에 충분하지 않습니다).
* 세 번째 네트워크는 논리 OR을 수행합니다.   
  뉴런 C는 뉴런 A 또는 뉴런 B가 활성화되거나 둘 다 활성화되면 활성화됩니다.
* 마지막으로, 뉴런 A가 항상 활성화되어 있으면 논리 NOT을 수행합니다.   
  뉴런 A가 활성화되어 있고 뉴런 B가 꺼져 있으면 뉴런 C가 활성화된다. 즉 뉴런 B가 꺼져있을 때 뉴런 C가 활성화되고 반대의 경우도 마찬가지입니다.

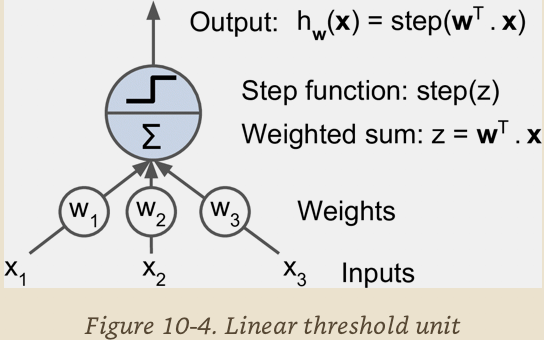
이러한 네트워크를 결합하여 복잡한 논리 표현식을 계산하는 방법을 쉽게 상상할 수 있습니다(이 장의 끝 부분에있는 실습 참조).

* 1. **The Perceptron**

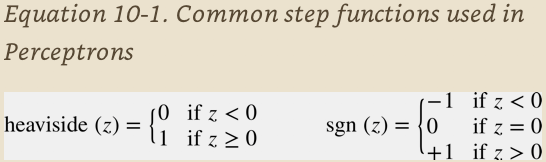
Perceptron은 Frank Rosenblatt가 1957 년에 발명 한 가장 단순한 ANN 아키텍처 중 하나입니다. 선형 임계 값 단위 (LTU)라고하는 약간 다른 인공 뉴런 (그림 10-4 참조)을 기반으로 합니다. 입력 및 출력은 이제 숫자가 되고 (on/off 대신) 각 입력 연결은 가중치와 연결됩니다.

LTU는 입력 (z = w1 x1 + w2 x2 + ⋯ + wn xn = wT · x)의 가중 합을 계산한 다음 그 합에 스텝 함수를 적용하여 결과를 출력합니다.

hw (x) = step = step (wT · x).



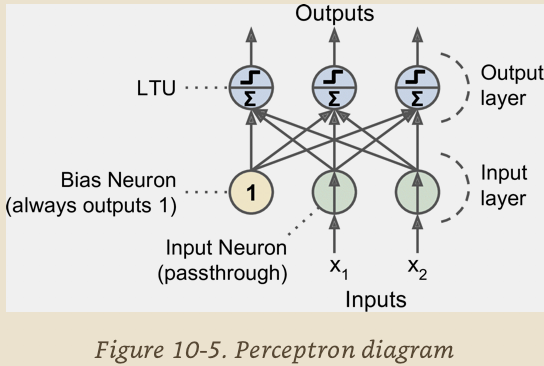
퍼셉트론에서 사용되는 가장 일반적인 단계 함수는 Heaviside 스텝 함수입니다 (식 10-1 참조). 때때로 sign 함수가 대신 사용됩니다.



단일 선형 이진 분류에 단일 LTU를 사용할 수 있습니다. 입력의 선형 조합을 계산하고 결과가 임계 값을 초과하는 경우 양수 클래스를 출력하거나 그렇지 않으면 (Logistic Regression 분류기 또는 선형 SVM과 같은) 음수 클래스를 출력합니다. 예를 들어, 단일 LTU를 사용하여 꽃잎의 길이와 너비를 기반으로 Iris 꽃을 분류 할 수 있습니다 (이전 장에서했던 것처럼 추가 바이어스 기능 x0 = 1 추가). LTU를 훈련한다는 것은 w0, w1 및 w2에 대한 올바른 값을 찾는 것을 의미합니다 (훈련 알고리즘은 곧 논의됩니다).

Perceptron은 LTU의 단일층으로 구성되며, 각 입력에는 모든 뉴런이 연결됩니다. 이러한 연결은 종종 입력 뉴런이라고하는 특별한 패스 스루 뉴런을 사용하여 표현됩니다. 입력 된 모든 입력을 출력합니다. 또한 추가 바이어스 특성이 일반적으로 추가됩니다 (x0 = 1). 이 바이어스 특성은 일반적으로 항상 1을 출력하는 바이어스 뉴런이라는 특별한 유형의 뉴런을 사용하여 표현됩니다.

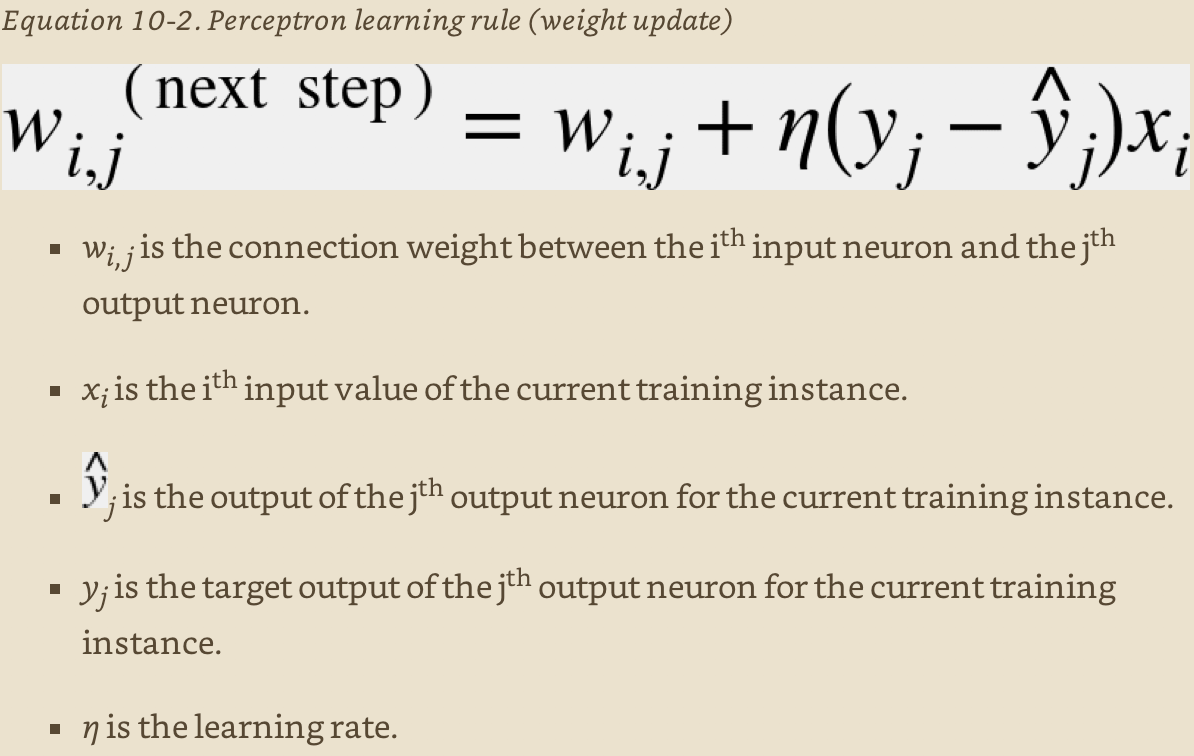
그림 10-5는 두 개의 입력과 세 개의 출력을 갖는 퍼셉트론을 나타냅니다. 이 Perceptron은 인스턴스를 동시에 세 가지 다른 이진 클래스로 분류 할 수 있으므로 다중 출력 분류기입니다.



그럼 퍼셉트론은 어떻게 트레이닝할까?

Frank Rosenblatt가 제안한 Perceptron 트레이닝 알고리즘은 Hebb’s rule에 크게 영향을 받았습니다. Donald Hebb는 생물학 뉴런이 종종 다른 뉴런을 유발할 때 이 두 뉴런 간의 연결이 강하게 성장한다고 1949 년에 출판 된 The Behavior of Organisation에서 제안했습니다. 이 아이디어는 나중에 Siegrid Löwel이 캐치 프레이즈로 요약했습니다 : "함께 타는 세포는 서로 연결됩니다." 이 규칙은 후에 Hebb’s rule (또는 Hebbian learning)으로 알려지게되었습니다. 즉, 동일한 출력을 가질 때마다 두 뉴런 사이의 연결 가중치가 증가합니다.

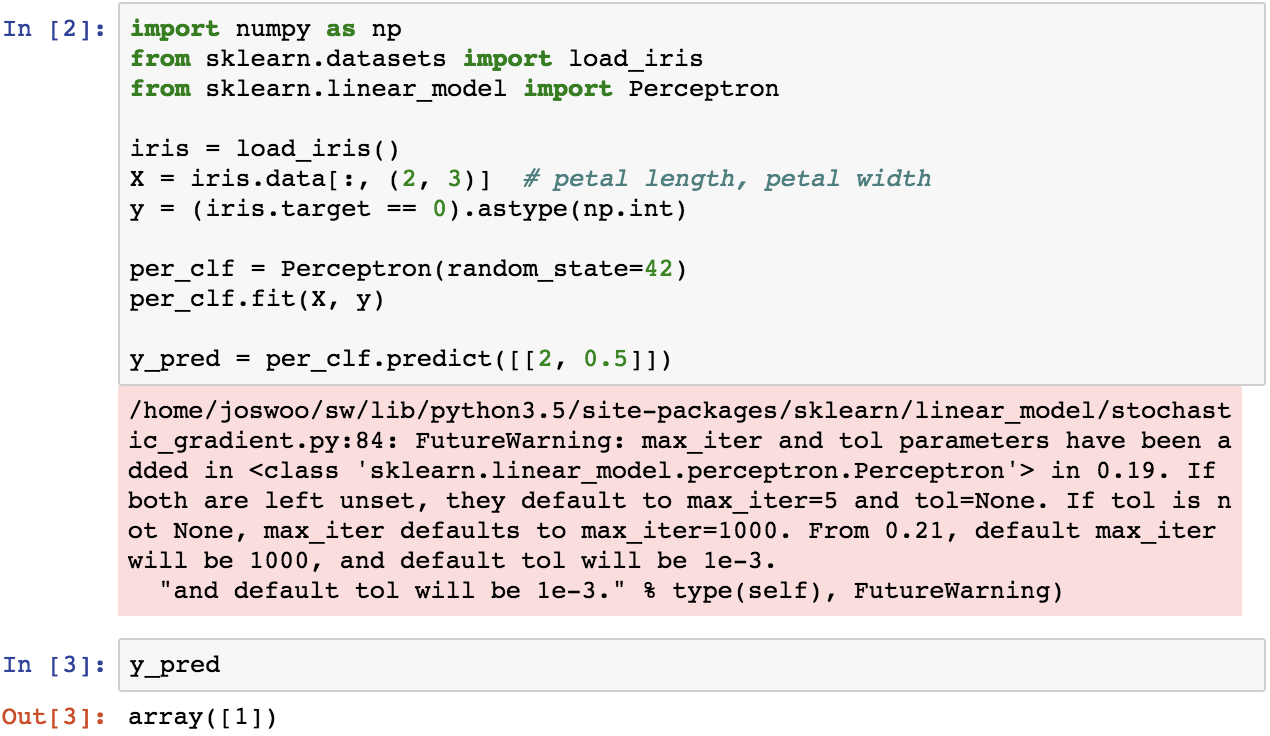
보다 구체적으로 말하면, 퍼셉트론은 한 번에 하나의 교육 인스턴스로 공급되며, 각 인스턴스에 대해 예측을합니다. 잘못된 예측을 산출 한 모든 출력 뉴런에 대해 올바른 예측에 기여한 입력으로부터 연결 가중치를 강화합니다. 규칙은 식 10-2와 같습니다.



각 출력 뉴런의 결정 경계는 선형이므로 Perceptron은 복잡한 패턴을 학습 할 수 없습니다 (Logistic Regression 분류 자와 유사 함). 그러나 트레이닝 인스턴스가 선형으로 분리 가능한 경우 Rosenblatt는 이 알고리즘이 솔루션에 수렴한다는 것을 입증했습니다. 이것을 퍼셉트론 수렴정리라고합니다.

Scikit-Learn은 단일 LTU 네트워크를 구현하는 Perceptron 클래스를 제공합니다.

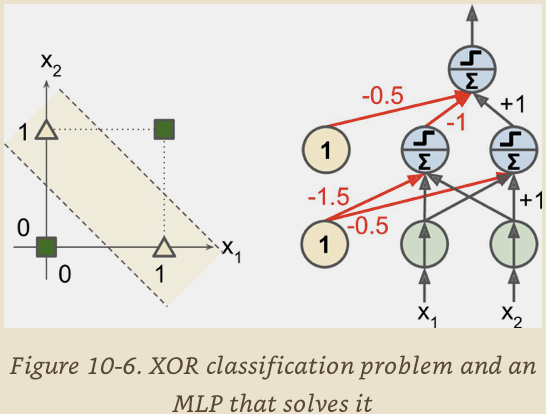
예를 들어 Iris 데이터 세트 (예 : 4 장에서 소개)에서 예상대로 사용할 수 있습니다.



Perceptron 학습 알고리즘이 Stochastic Gradient Descent와 매우 유사하다는 것을 알고있을 것입니다. 실제로 Scikit-Learn의 Perceptron 클래스는 loss = "perceptron", learning\_rate = "constant", eta0 = 1 (학습 속도) 및 penalty = None (정규화 없음)과 같은 하이퍼 매개 변수와 함께 SGDClassifier를 사용하는 것과 같습니다.

로지스틱 회귀 분류기와 달리, 퍼셉트론은 클래스 확률을 출력하지 않습니다. 오히려, 이것은 단지 하드 임계값을 기반으로 예측을합니다. 이것은 Perceptrons 보다 Logistic Regression을 선호하는 좋은 이유 중 하나입니다.

Perceptrons라는 제목의 1969년 논문에서 Marvin Minsky와 Seymour Papert는 Perceptrons의 심각한 약점, 특히 사소한 문제 (Exclusive OR (XOR) 분류 문제를 해결할 수 없다는 사실을 강조했다. 그림 10-6).



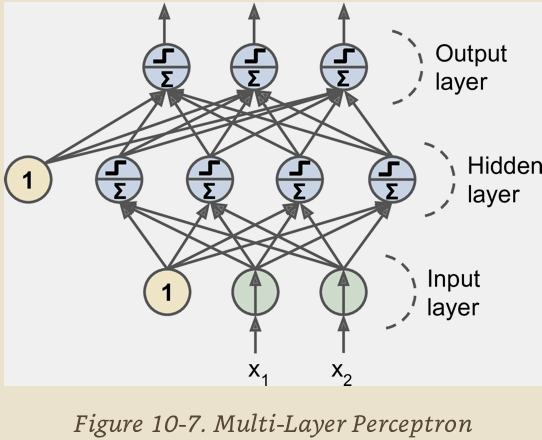
물론 이것은 로지스틱 회귀 분류기와 같은 다른 선형 분류 모델에서도 마찬가지이지만 연구원은 퍼셉트론에서 더 많은 것을 기대했으며 실망감이 컸습니다. 결과적으로 많은 연구자들이 연결주의를 완전히 포기했습니다 (즉, 연구 논리 네트워크, 문제 해결 및 검색과 같은 상위 수준의 문제에 유리합니다.

그러나 여러 가지 퍼셉트론을 스태킹하면 퍼셉트론의 한계점 중 일부를 제거할 수 있습니다. 결과 ANN을 MLP (Multi-Layer Perceptron)라고합니다. 특히 MLP는 입력의 각 조합에 대해 그림 10-6의 오른쪽에 표시된 MLP의 출력을 계산하여 확인할 수 있으므로 XOR 문제를 해결할 수 있습니다. 입력 ((0, 0) 또는 (1, 1)) 네트워크는 0을 출력하고 입력 (0, 1) 또는 (1, 0)을 출력하면 1을 출력합니다.

* 1. **Multi-Layer Perceptron and Backpropagation**

MLP는 하나의 패스 스루 입력 레이어, 하나 이상의 LTU 레이어(숨겨진 레이어) 및 LTU의 최종 레이어(출력 레이어)로 구성됩니다(그림 10-7 참조). 출력 레이어를 제외한 모든 레이어는 바이어스 뉴런을 포함하며 다음 레이어에 완전히 연결됩니다.

ANN에 두 개 이상의 숨겨진 레이어가 있는 경우 DNN (Deep Neural Network)라고 합니다.



수년 동안 연구자들은 성공하지 못하고 MLP를 훈련시키는 방법을 찾는데 어려움을 겪었다.

그러나 1986 년 D. E. Rumelhart et al. 은 Backpropagation 학습 알고리즘을 도입한 groundbreaking article8를 발표했습니다. 오늘날 우리는 reverse-mode autodiff (Gradient Descent는 4 장, autodiff는 9 장에서 논의되었습니다)를 사용하여 Gradient Descent로 설명 할 것입니다. 각 교육에 대해

각 트레이닝 인스턴스에 대해 알고리즘은 네트워크에 피드를 제공하고 각 연속 레이어의 모든 뉴런 출력을 계산합니다(순방향 전달).

* 그런 다음 네트워크의 출력 오류(즉, 원하는 출력과 네트워크의 실제 출력 간의 차이)를 측정한다.
* 마지막 숨겨진 계층부터 이전의 숨겨진 레이어의 각 뉴런에서 발생한 오류 기여도가 얼마나되는지 측정합니다. 알고리즘이 입력 레이어에 도달 할 때까지 계속됩니다. (역방향 전달)
* 오류를 줄이기 위해서 역전파 알고리즘의 마지막 단계는 앞에서 측정 한 오류 기울기를 사용하여 네트워크의 모든 연결 가중치에 대한 Gradient Descent 입니다.

이 알고리즘이 제대로 작동하려면 작성자가 MLP의 구조를 크게 변경했습니다.

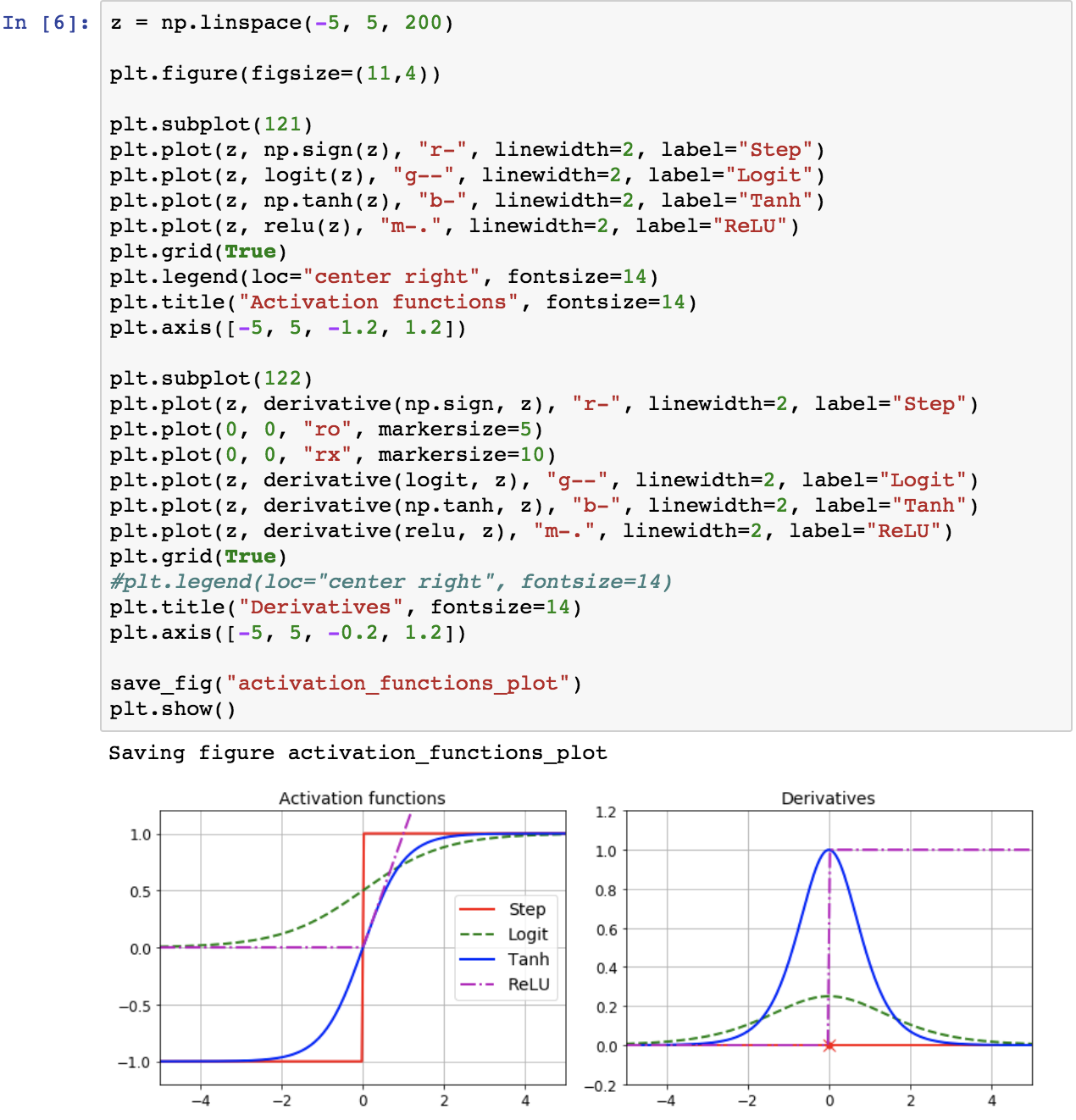
단계 함수를 Logistic 함수 σ (z) = 1 / (1 + exp (-z))로 대체했습니다.

계단 함수에는 평평한 세그먼트로만 구성되기 때문에 그라디언트 하강은 평평한 표면에서 움직일 수 없어서 대체가 필수적이었다.

반면, Logistic 함수는 모든 곳에서 잘 정의된 0이 아닌 파생 값을 가지므로 Gradient Descent가 모든 단계에서 진행. 역전파 알고리즘은 물류 기능 대신 다른 활성화 기능과 함께 사용될 수 있습니다. 두 가지 다른 인기있는 활성화 함수은 다음과 같습니다.

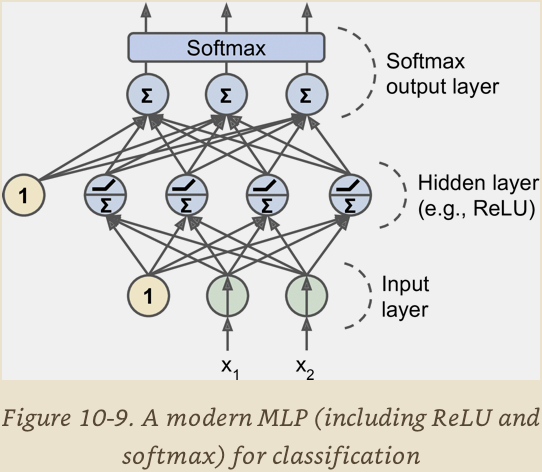
* **쌍곡선 탄젠트 함수 tanh (z) = 2σ (2z) – 1**  
  물류 기능과 마찬가지로 S 자형이며 연속적이며 차별화가 가능하지만 출력 값의 범위는 물류 기능의 경우 0에서 1 대신 -1에서 1까지이며 각 레이어의 출력을 더 많이 또는 훈련 시작시 덜 정규화 (즉, 0을 중심으로 함). 이는 수렴 속도를 높이는데 종종 도움이됩니다.
* **ReLU 기능 (9 장에서 소개)**  
  ReLU (z) = max (0, z). 그것은 연속적이지만 불행하게도 z = 0에서 구별 할 수 없습니다 (기울기가 갑자기 변하기 때문에 그라디언트 하강이 바운스 될 수 있습니다). 그러나 실제로는 매우 잘 작동하며 계산 속도가 빠르다는 이점이 있습니다. 무엇보다 최대 출력 값을 갖고 있지 않다는 사실은 그라디언트 하강 동안 일부 문제를 줄이는데 도움이됩니다 (11 장에서 다시 설명 할 것입니다).

위의 활성화 함수와 도함수는 아래와 같다.



MLP는 종종 분류에 사용되며 각 출력은 다른 바이너리 클래스 (spam/ham, urgent/not-urgent)에 해당합니다. 클래스가 배타적이면 (숫자 이미지 분류의 경우 클래스 0-9) 출력 레이어는 일반적으로 개별 활성화 함수를 softmax 함수로 대체하여 수정됩니다 (그림 10-9 참조).

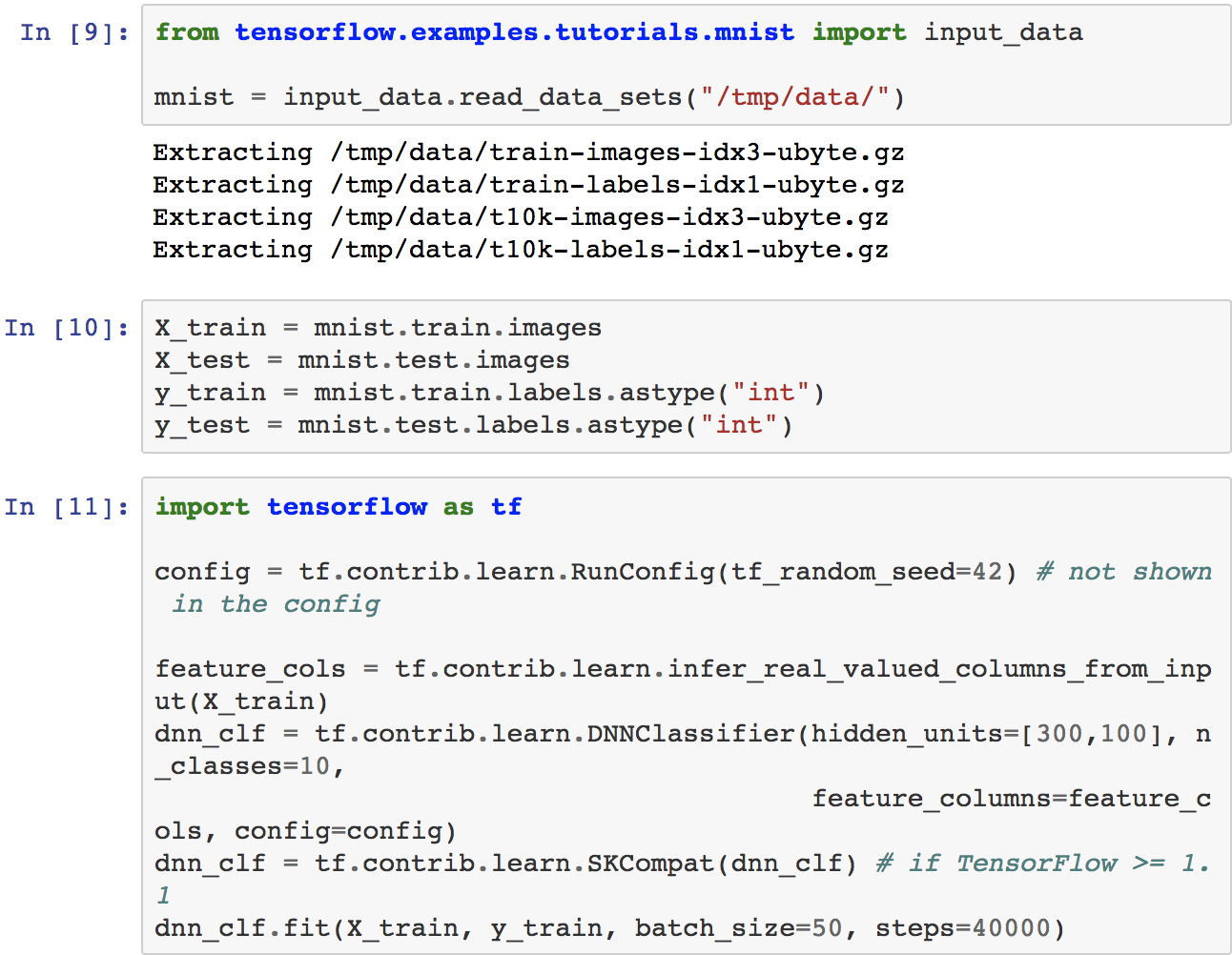
각 뉴런의 출력은 해당 클래스의 예상 확률에 해당합니다. 신호는 입력에서 출력으로 한 방향으로만 흐릅니다. 따라서 이 구조는 FNN (feedforward neural network)의 예입니다.



생물학적 뉴런은 대략 S(sigmod) 자 형태의 활성화 기능을 구현하는 것으로 보이므로 연구원들은 오랫동안 S 자 형태의 기능을 고수했다. 그러나 ReLU 활성화 기능은 일반적으로 ANN에서 더 잘 작동하는 것으로 나타났습니다. 이것은 생물학적 유추가 오도된 경우 중 하나입니다.

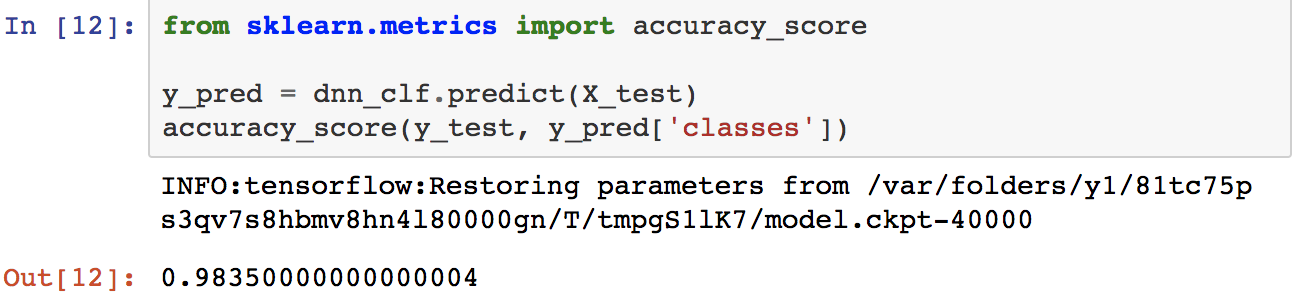
1. **Training an MLP with TensorFlow’s High-Level API**

TensorFlow를 사용하여 MLP를 교육하는 가장 간단한 방법은 Scikit-Learn 호환 API를 제공하는 고급 API TF.Learn을 사용하는 것입니다. DNNClassifier 클래스를 사용하면 숨겨진 레이어를 여러 개 사용하여 심층 신경 네트워크를 쉽게 학습 할 수 있고 예상되는 클래스 확률을 출력 할 softmax 출력 레이어를 만들 수 있습니다. 예를 들어, 다음 코드는 두 개의 숨겨진 레이어 (하나는 300 개의 뉴런이 있고 다른 하나는 100 개의 뉴런)와 10 개의 뉴런이있는 softmax 출력 레이어로 분류되도록 DNN을 교육합니다.



이 코드는 먼저 교육 세트에서 실제 가치가 있는 열 집합을 만듭니다(categorical columns 같은 다른 유형의 열을 사용할 수 있음). 그런 다음 DNNClassifier를 만들고 Scikit-Learn compatibility helper로 묶습니다. 마지막으로 50 개의 인스턴스를 사용하여 40,000 개의 트레이닝 반복을 실행합니다.

이 코드를 MNIST 데이터 세트에서 실행하면 (예 : Scikit-Learn의 StandardScaler를 사용하여 크기를 조정 한 후) 실제로 테스트 세트에서 약 98.2 %의 정확도를 달성하는 모델을 얻을 수 있습니다. 3 장에서 훈련한 모델들보다 낫습니다.



DNNClassifier 클래스는 ReLU 활성화 함수 (activation\_fn 하이퍼 매개 변수를 설정하여 변경할 수 있음)를 기반으로 모든 뉴런 레이어를 만듭니다. 출력 레이어는 softmax 함수에 의존하고 비용 함수는 교차 엔트로피 (4 장에서 소개 함)입니다.

1. **Training a DNN Using Plain TensorFlow**

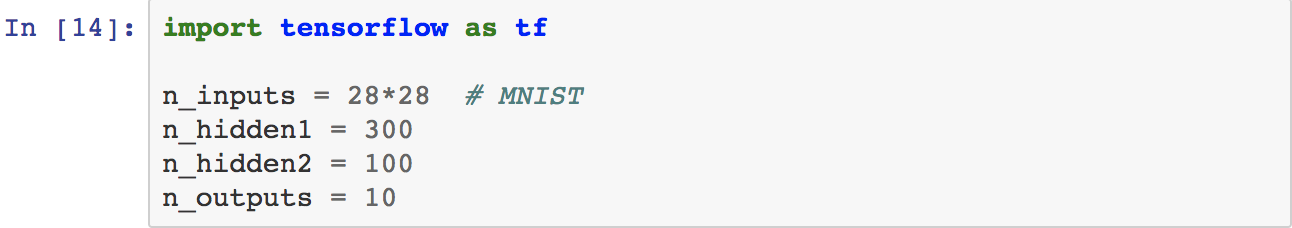
네트워크 구조를보다 잘 제어하려면 TensorFlow의 하위 레벨 Python API (9 장에서 소개 함)를 사용하는 것이 좋습니다. 이 섹션에서는 이 API를 사용하기 전과 동일한 모델을 만들고 MNIST 데이터 세트에서 Mini-batch Gradient Descent를 구현합니다.

첫 번째 단계는 TensorFlow 그래프를 작성하는 구축 단계입니다.

두 번째 단계는 실행 단계입니다. 여기서 실제로 그래프를 실행하여 모델을 교육합니다.

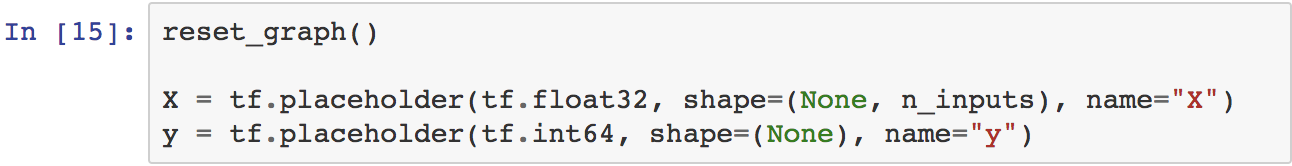
* 1. **Construction Phase**

먼저 tensorflow 라이브러리를 가져와야합니다. 그런 다음 입력 및 출력수를 지정하고 각 계층에서 숨겨진 뉴런 수를 설정해야합니다.



다음으로 9 장에서했던 것처럼 자리 placeholder 노드를 사용하여 교육 데이터와 대상을 나타낼 수 있습니다. X의 모양은 부분적으로 만 정의됩니다.

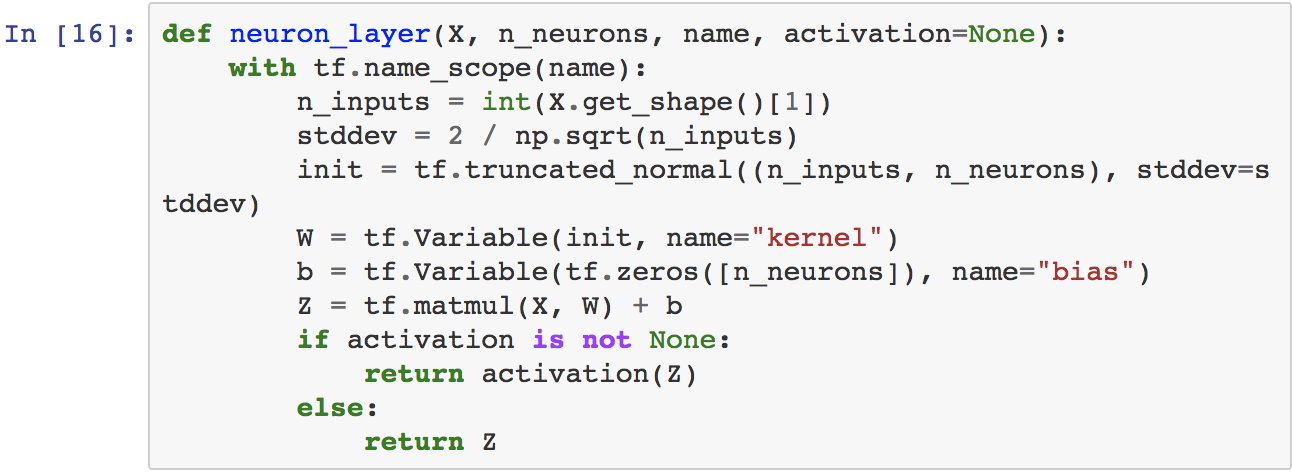
첫 번째 차원의 인스턴스와 두 번째 차원의 차원을 가진 2차원 텐서(즉, 행렬)가 될 것이고, 피쳐의 수는 28 x 28이 될 것입니다. 하지만 각 훈련 배치에 몇 개의 인스턴스가 포함되는지 아직 알 수 없습니다. 그래서 X의 모양은 (None, n\_inputs)입니다. 마찬가지로 y는 인스턴스 당 하나의 엔트리를 가진 1 차원 텐서가 될 것이지만 이 시점에서 트레이닝 배치의 크기를 알 수 없기 때문에 shape=(none)입니다.



이제 실제 신경망을 만들어 보겠습니다.

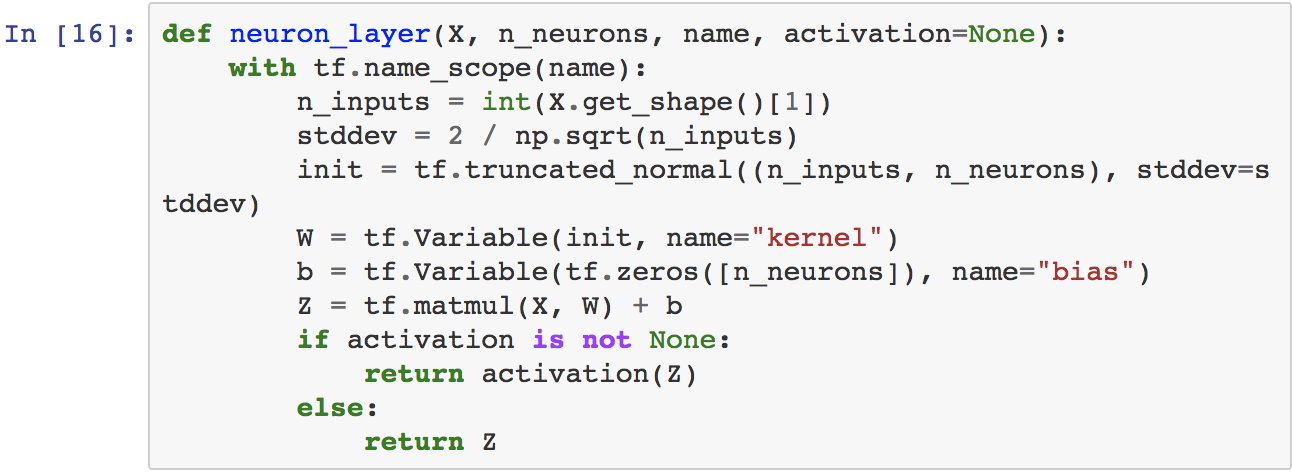
placeholder X는 입력 레이어로 작동합니다. 실행 단계에서 한 번에 하나의 트레이닝 배치로 대체됩니다 (교육 배치의 모든 인스턴스는 신경 네트워크에 의해 동시에 처리됩니다).

이제 두 개의 숨겨진 레이어와 출력 레이어를 만들어야 합니다. 두 개의 숨겨진 레이어는 거의 동일합니다. 연결되는 입력과 포함된 뉴런의 수에 따라 다릅니다. 출력 레이어도 매우 유사하지만 ReLU 활성화 기능 대신 softmax 활성화 기능을 사용합니다. 한 번에 하나의 레이어를 만드는데 사용할 neuron\_layer () 함수를 만듭니다. 입력, 뉴런 수, 활성화 함수 및 레이어 이름을 지정하는 매개 변수가 필요합니다.



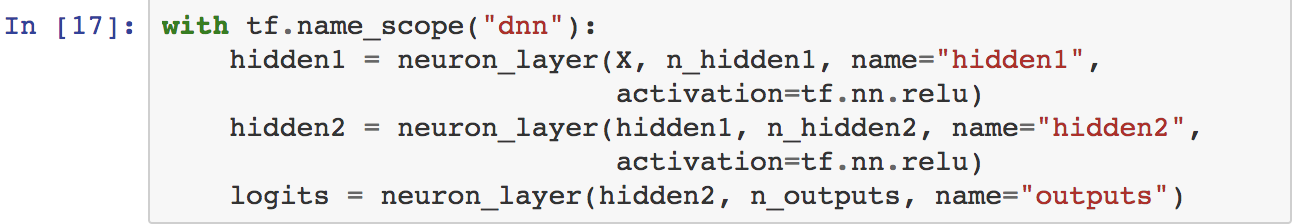
이 코드를 한 줄씩 살펴 보겠습니다.

1. 먼저 레이어 이름을 사용하여 이름 범위를 만듭니다.이 레이어에는이 뉴런 레이어에 대한 모든 계산 노드가 포함됩니다. 선택 사항이지만 TensorBoard의 노드가 잘 구성되어 있으면 그래프가 훨씬 더 멋지게 보입니다.
2. 다음으로 입력 행렬의 모양을 찾고 두 번째 차원의 크기를 가져 와서 입력 수를 얻습니다 (첫 번째 차원은 인스턴스 용입니다).
3. 다음 세 행(W까지)은 가중치 행렬 (종종 계층의 커널이라고 함)을 보유 할 W 변수를 작성합니다. 각 입력과 각 뉴런 사이의 모든 연결 가중치를 포함하는 2D 텐서입니다. 따라서 그 모양은 (n\_inputs, n\_neurons)가됩니다. 그것은 표준 편차가 truncated10 normal (Gaussian) 분포를 사용하여 무작위로 초기화됩니다. 이 특정 표준 편차를 사용하면 알고리즘이 훨씬 빨리 수렴하는 데 도움이됩니다 (11 장에서 더 자세히 논의 할 것입니다; 이는 효율성에 엄청난 영향을 미친 신경망에 대한 작은 조정 중 하나입니다). Gradient Descent 알고리즘이 깨뜨릴 수없는 대칭을 피하기 위해 모든 숨겨진 레이어에 대해 연결 가중치를 임의로 초기화하는 것이 중요합니다.



1. 다음 줄은 바이너리에 대한 b 변수를 만듭니다.이 변수는 뉴런 당 하나의 바이어스 매개 변수와 함께 0으로 초기화됩니다 (이 경우 대칭 문제 없음).
2. 그런 다음 Z = X / W + b를 계산하기위한 하위 그래프를 만듭니다. 이 벡터화 된 구현은 한 번에 배치의 모든 인스턴스에 대해 입력의 가중치 합계와 레이어의 각 뉴런에 대한 바이어스 조건을 효율적으로 계산합니다. 동일한 수의 열 (XW)을 가진 2D 행렬에 1D 배열 (b)을 추가하면 행렬의 모든 행에 1D 배열이 추가됩니다.이를 방송이라고합니다.
3. 마지막으로, tf.nn.relu (즉, max (0, Z))와 같은 활성화 매개 변수가 제공되면 코드는 활성화 (Z)를 반환하고 그렇지 않으면 Z를 반환합니다.

Deep neural network의 뉴런 레이어를 만드는 함수가 있습니다.



첫 번째 숨겨진 레이어는 X를 입력으로 사용합니다.

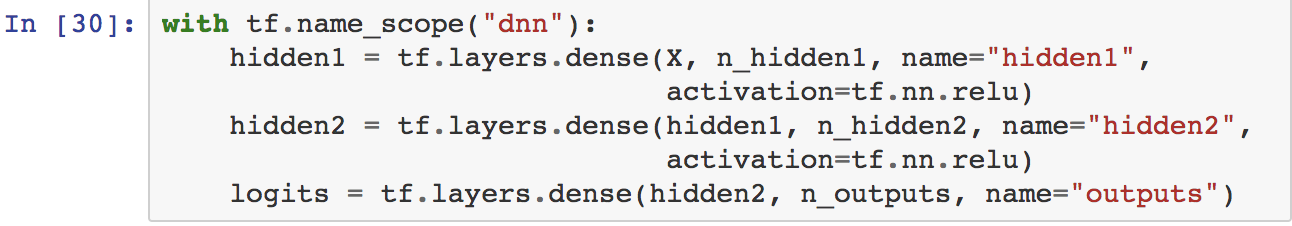
두 번째는 첫 번째 숨겨진 레이어의 출력을 입력으로 사용합니다.

출력 레이어는 두 번째 숨겨진 레이어의 출력을 입력으로 사용합니다.

다시 한번 우리는 명확성을 위해 name\_scope를 사용했습니다. 또한 logits는 softmax 활성화 함수를 거치기전에 신경망의 출력입니다. 최적화를 위해 나중에 softmax 계산을 처리할 것입니다.

예상대로 TensorFlow에는 표준 신경망 레이어를 만드는데 유용한 많은 기능이 제공되므로 방금 수행한 것처럼 자신의 neuron\_layer () 함수를 정의할 필요가 없는 경우가 종종 있습니다. 예를 들어, TensorFlow의 tf.layers.dense () 함수는 모든 입력이 레이어의 모든 신경에 연결되는 완전히 연결된 레이어를 생성합니다.

적절한 초기화 전략을 사용하여 kernel(weight)및 bias(bias)를 작성하고 activation 인수를 사용하여 활성화 함수를 설정할 수 있습니다. 11 장에서 볼 수 있듯이 정규화 매개 변수도 지원합니다. 앞의 코드에서 neuron\_layer () 함수 대신 dense () 함수를 사용하도록 조정 해 봅시다. dnn 구성 섹션을 다음 코드로 바꾸기 만하면됩니다.

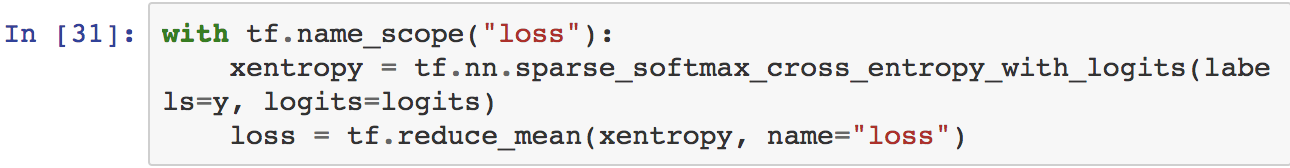


이제 신경 네트워크 모델을 사용할 준비가 되었으므로, 우리는 이를 훈련시키는데 사용할 비용 함수를 정의해야합니다. 4 장의 Softmax Regression과 마찬가지로 교차 엔트로피를 사용합니다. 앞서 논의한 것처럼 교차 엔트로피는 목표 클래스에 대한 낮은 확률을 추정하는 모델에 불이익을 줄 것입니다.

TensorFlow는 교차 엔트로피를 계산하는 몇 가지 기능을 제공합니다.

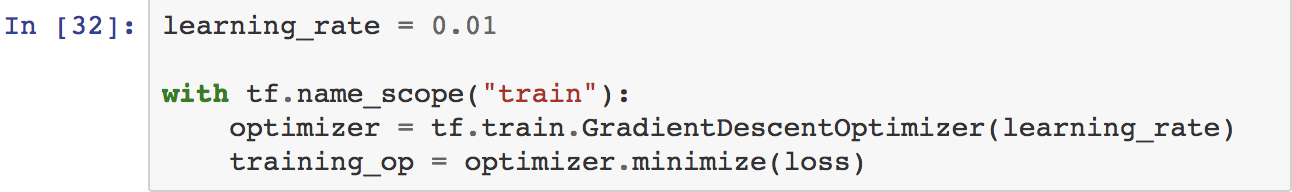
sparse\_softmax\_cross\_entropy\_with\_logits ()는 “logits"(즉, softmax 활성화 함수를 거치기 전에 네트워크의 출력)를 기준으로 교차 엔트로피를 계산하며 0부터 클래스에서 1을 뺀 숫자 까지의 정수 형식의 레이블이 필요합니다 (우리의 경우 0에서 9까지).

이것은 각 인스턴스에 대해 교차 엔트로피를 포함하는 1D 텐서를 제공합니다. 그런 다음 TensorFlow의 reduce\_mean () 함수를 사용하여 모든 인스턴스에 대한 평균 교차 엔트로피를 계산할 수 있습니다



*[sparse\_softmax\_cross\_entropy\_with\_logits () 함수는 softmax 활성화 함수를 적용한 다음 교차 엔트로피를 계산하는 것과 동일하지만보다 효율적이며 모서리의 경우를 적절히 처리합니다. 로그가 클 경우 부동 소수점 올림 오류로 인해 softmax 출력이 발생할 수 있습니다 이 경우에는 교차 엔트로피 방정식에 음의 무한대와 같은 log (0) 항이 포함됩니다. sparse\_softmax\_cross\_ entropy\_with\_logits () 함수는 log (ε)를 계산하여이 문제를 해결합니다. 여기서 ε은 작은 양수입니다. 이것이 softmax 활성화 기능을 이전에 적용하지 않은 이유입니다. softmax\_cross\_ entropy\_with\_logits ()라는 또 다른 함수가 있습니다. 이 함수는 1 핫 벡터 형태로 레이블을 취합니다 (0에서 클래스 빼기 -1까지의 int 대신).]*

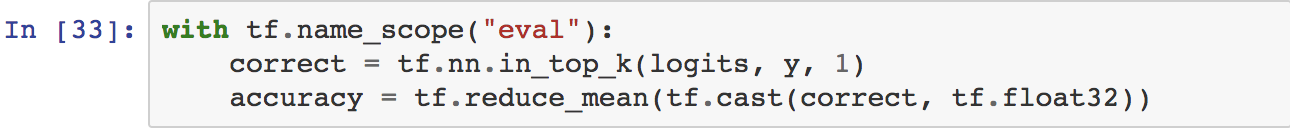
우리는 신경망 모델을 가지고 있고, 우리는 비용 함수를 가지고 있습니다. 그리고 이제 비용 함수를 최소화하기 위해 모델 매개 변수를 조정할 GradientDescentOptimizer를 정의해야 합니다.



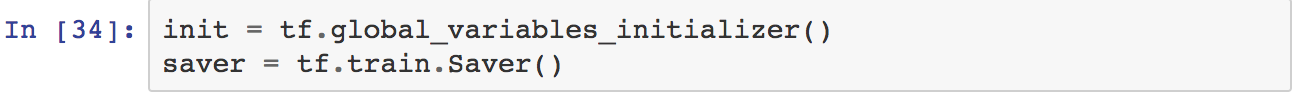
구축 단계의 마지막 중요한 단계는 모델을 평가하는 방법을 지정하는 것입니다.

우리는 단순히 성능 척도로서 정확도를 사용합니다. 각 인스턴스에 대해 가장 높은 logit이 목표 클래스에 해당하는지 여부를 확인하여 신경망의 예측이 올바른지 확인합니다. 이를 위해 in\_top\_k () 함수를 사용할 수 있습니다. 부울 값으로 가득 찬 1차원 텐서를 반환하므로 이 부울을 float형으로 캐스팅한 다음 평균을 계산해야합니다.

이렇게하면 네트워크의 전체적인 정확성을 얻을 수 있습니다.



평소처럼 모든 변수를 초기화하는 노드를 만들어야하며 훈련 된 모델 매개 변수를 디스크에 저장하는 Saver도 만듭니다.



이것으로 constriction 단계가 끝납니다. 이것은 40 줄 미만의 코드 였지만

* 입력 및 대상에 대한 placeholder를 만들었다.
* 신경층을 만드는 함수를 만들었다.
* DNN을 만드는 데 사용한 다음 비용 함수를 정의했다.
* 최적화 도구를 만들었다.
* 성능 측정 값을 정의했습니다.

이제 실행 단계로 넘어갑니다.

* 1. **Execution Phase**

이 부분은 훨씬 더 짧고 간단합니다. 먼저 MNIST를 로드해 봅시다. 우리는 이전 장에서했던 것처럼 Scikit-Learn를 사용할 수 있지만 TensorFlow는 데이터를 가져오고 (0에서 1 사이에서) 크기를 조정하고 셔플 링하며 하나의 미니 배치를로드하는 간단한 함수를 제공하는 자체 도우미를 제공합니다. 시각. 또한 데이터는 이미 교육 세트 (55,000 개 인스턴스), 유효성 검사 세트 (5,000 개 인스턴스) 및 테스트 세트 (10,000 개 인스턴스)로 분할되어 있습니다. 이 도우미를 사용합시다.



이제 실행하려는 epoch의 수와 미니 배치의 크기를 정의합니다.

이제 모델을 훈련 할 수 있습니다.



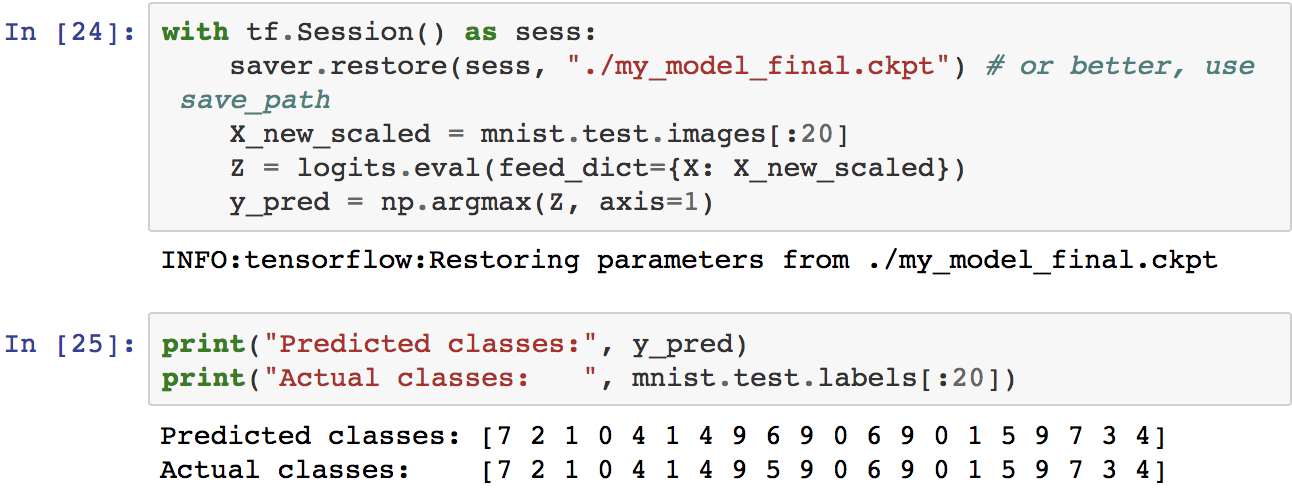
이 코드는 TensorFlow 세션을 열고 모든 변수를 초기화하는 init 노드를 실행합니다.

그런 다음 기본 트레이닝 루프를 실행합니다. 각 epoch에서 코드는 교육 세트 크기에 해당하는 여러 가지 미니 배치를 반복합니다. 각 미니 배치는 next\_batch () 메소드를 통해 가져온 후 코드는 현재 훈련 작업을 실행하고 현재 미니 배치 입력 데이터와 대상을 제공합니다.

다음으로, 각 epoch의 끝에서 코드는 마지막 미니 배치 및 전체 유효성 검사 집합에서 모델을 평가하고 결과를 인쇄합니다. 마지막으로 모델 매개 변수가 디스크에 저장됩니다.

* 1. **Using the Neural Network**

이제 신경 회로망이 트레이닝 되었으므로 이를 사용하여 예측을 할 수 있습니다. 그렇게 하기 위해 동일한 구성 단계를 재사용 할 수 있지만 다음과 같이 실행 단계를 변경하십시오.



* 코드를 보면 먼저 디스크에서 모델 매개 변수를 로드합니다.
* 그런 다음 분류 할 새 이미지를 로드합니다. 교육 데이터의 경우와 동일한 기능 확장을 적용해야합니다 (이 경우 크기를 0에서 1로 조정).
* 그런 다음 코드는 logits 노드를 평가합니다. 모든 예상 클래스 확률을 알고 싶다면 softmax () 함수를 logits에 적용해야 하지만 argmax () 함수는 클래스를 예측하려는 경우 가장 높은 logit 값을 가진 클래스를 선택할 수 있습니다.

1. **Fine-Tuning Neural Network Hyperparameters**

신경 네트워크의 유연성 또한 주요 단점 중 하나입니다.

조정할 수 있는 많은 하이퍼 파라미터가 있습니다. 상상할 수 있는 네트워크 토폴로지(뉴런이 어떻게 상호 연결되는지)를 사용할 수 있을 뿐만 아니라 간단한 MLP에서도 레이어 수, 레이어 당 뉴런 수, 각 레이어에서 사용할 활성화 함수 유형, 가중치 등을 변경할 수 있습니다

하이퍼 매개 변수의 조합이 작업에 가장 적합한 방법을 어떻게 알 수 있습니까?

물론 이전 장에서 했던 것처럼 교차 검증(cross-validation)과 함께 그리드 검색(grid search)을 사용하여 올바른 하이퍼 매개 변수를 찾을 수 있지만 튜닝 할 하이퍼 매개 변수가 많고 대규모 데이터 집합에서 신경망을 학습하는 데 많은 시간이 걸립니다.

2 장에서 설명한 것처럼 무작위 검색(randomized search)을 사용하는 것이 훨씬 더 좋습니다. 또 다른 옵션은 Oscar와 같은 도구를 사용하는 것입니다. 이 도구는 더 복잡한 알고리즘을 구현하여 훌륭한 하이퍼 매개 변수 세트를 빠르게 찾을 수 있도록 도와줍니다.

각 하이퍼 매개 변수에 대해 합리적인 값이 무엇인지 생각하면 검색 공간을 제한 할 수 있습니다. 숨겨진 레이어의 수부터 시작해 보겠습니다.

* 1. **Number of Hidden Layers**

많은 문제가 발생하면 하나의 숨겨진 레이어로 시작하면 합리적인 결과를 얻을 수 있습니다. 실제로 하나의 숨겨진 레이어가있는 MLP가 충분한 뉴런을 제공한다면 가장 복잡한 기능도 모델링 할 수 있다는 사실이 실제로 입증되었습니다. 그러나 깊은 네트워크는 얕은 네트워크보다 매개 변수 효율성이 훨씬 높다는 사실이 있습니다. 또, 얕은 네트워크 보다 기하 급수적으로 적은 뉴런을 사용하여 복잡한 함수를 모델링 할 수 있으므로 훈련이 훨씬 빨라집니다.

<이유를 이해하려면 일부 드로잉 소프트웨어를 사용하여 포리스트를 그리도록 요청 받았지만 복사 / 붙여 넣기를 사용하는 것은 금지되어 있습니다. 각 트리를 개별적으로 그려야하고, 분기마다 분기하고 잎마다 잎을 그려야합니다. 하나의 잎을 그릴 수 있다면 그것을 복사하여 붙여 넣기를하고 그 지점을 복사 / 붙여 넣기하여 트리를 만들고 마지막으로 이 트리를 복사하여 붙여 넣어 트리를 만들면 곧 마칠 수 있습니다.>

실제 데이터는 이러한 계층적 방식으로 구조화되는 경우가 많으며 DNN은 자동으로 이 사실을 활용합니다. 낮은 숨겨진 레이어 모델 (예 : 다양한 모양과 방향의 선분) 하위 구조 모델 중간 레벨 구조 (예 : 사각형, 원형)를 모델링하고 최상위 숨겨진 레이어와 출력 레이어는 이러한 중간 구조를 결합하여 상위 수준 구조 (예 : 얼굴)를 모델링합니다.

이 계층적 구조는 DNN이보다 신속하게 좋은 솔루션으로 수렴 할 수 있을 뿐만 아니라 새로운 데이터 세트로 일반화 할 수있는 능력을 향상시킵니다.

예를 들어 사진에서 얼굴을 인식하도록 모델을 이미 훈련했으며 새 신경망을 학습하여 헤어 스타일을 인식하려는 경우 첫 번째 네트워크의 하위 레이어를 다시 사용하여 교육을 시작할 수 있습니다. 새로운 신경망의 처음 몇 개의 레이어의 가중치와 바이어스를 무작위로 초기화하는 대신, 첫 번째 네트워크의 하위 레이어의 가중치와 바이어스 값으로 초기화 할 수 있습니다. 이 방법으로 네트워크는 대부분의 그림에서 발생하는 모든 하위 구조를 처음부터 배우지 않아도 됩니다. 상위 구조(예 : 헤어 스타일)만 학습하면 됩니다.

요약하면, 많은 문제에 대해 하나 또는 두 개의 숨겨진 레이어로 시작할 수 있습니다.

예를 들어, 수백 개의 뉴런이있는 하나의 숨겨진 레이어를 사용하여 MNIST 데이터 세트에서 97 % 이상의 정확도에 쉽게 도달 할 수 있으며 대략 동일한 양의 교육 시간에 동일한 총량의 뉴런으로 2 개의 숨겨진 레이어를 사용하여 98 % 이상의 정확도를 얻을 수 있습니다.

좀 더 복잡한 문제의 경우, 트레이닝 세트를 오버피팅될때까지 숨겨진 레이어의 수를 점차적으로 늘릴 수 있습니다. 큰 이미지 분류 또는 음성 인식과 같은 매우 복잡한 작업에는 일반적으로 수십 개의 레이어가있는 네트워크가 필요합니다. 그러나 그러한 네트워크를 처음부터 교육 할 필요는 거의 없습니다. 유사한 작업을 수행하는 미리 훈련 된 최첨단 네트워크의 일부를 재사용하는 것이 훨씬 더 일반적입니다. 교육은 훨씬 빨라지고 훨씬 적은 데이터가 필요합니다 (11 장에서 이에 대해 논의 할 것입니다).

* 1. **Number of Neurons per Hidden Layer**

입력 및 출력 레이어의 뉴런 수는 작업에 필요한 입력 및 출력 유형에 따라 결정됩니다.

예를 들어, MNIST 작업에는 28 x 28 = 784 입력 뉴런과 10 출력 뉴런이 필요합니다.

숨겨진 레이어의 경우 일반적으로 각 레이어에서 더 적은 수의 뉴런을 사용하여 깔때기를 형성하도록 크기를 조정하는 것이 일반적입니다. 그 이유는 많은 낮은 수준의 기능이 훨씬 적은 수의 높은 수준의 기능으로 통합 될 수 있다는 것입니다.

예를 들어, MNIST의 일반적인 신경망은 두 개의 숨겨진 레이어를 가질 수 있습니다. 첫 번째 레이어는 300 개의 뉴런과 두 번째 레이어는 100입니다. 그러나 이 방법은 지금처럼 일반적이지 않으며 모든 숨겨진 레이어에 동일한 크기를 사용할 수 있습니다. 예를 들어 150 개의 뉴런이있는 모든 숨겨진 레이어는 레이어 당 하나가 아닌 하나의 하이퍼 파라미터로 조정됩니다. 레이어 수와 마찬가지로 네트워크가 지나치게 시작될 때까지 점차적으로 뉴런의 수를 늘릴 수 있습니다. 일반적으로 레이어당 뉴런의 수보다 레이어 수를 늘려서 더 많은 수익을 올릴 수 있습니다. 불행히도, 당신이 볼 수 있듯이, 완벽한 양의 뉴런을 찾는 것은 여전히 ​​다소 검은 예술입니다.(트릭같다)

더 간단한 접근법은 실제로 필요한 것보다 더 많은 레이어와 뉴런을 가진 모델을 선택한 다음 조기 정지를 사용하여 오버 피팅을 방지합니다. 이것은 "stretch pants "접근 방식으로 불립니다. 시간을 낭비하는 대신 바지를 찾고 크기를 맞추기 위해 큰 사이즈의 스트레칭 바지를 사용하십시오.

* 1. **Activation Functions**

대부분의 경우 숨겨진 레이어에서 ReLU 활성화 함수를 사용할 수 있습니다. 다른 활성화 함수보다 계산 속도가 약간 빠르며 Gradient Descent 은 큰 입력 값에 대해 포화 상태가 아니므로 고원(plateaus)에 그리 걸리지 않습니다.

출력 레이어의 경우 softmax 활성화 함수는 일반적으로 클래스가 상호 배타적인 경우 분류 작업에 적합한 선택입니다. 상호 배타적이지 않을 때 (또는 클래스가 두 개인 경우) 일반적으로 물류 기능을 사용하려고합니다. 회귀 작업의 경우 출력 레이어에 활성화 기능을 전혀 사용하지 않아도됩니다.

이것으로 인공 신경망에 대한 소개를 마칩니다. 다음 장에서는 매우 깊이있는 네트워크를 훈련하고 여러 서버와 GPU에 걸쳐 교육을 배포하는 기술에 대해 설명합니다. 그런 다음 몇 가지 다른 대중적인 신경 네트워크 아키텍처, 즉 길쌈 신경 네트워크, 반복 신경 네트워크 및 자동 인코딩을 살펴 봅니다.