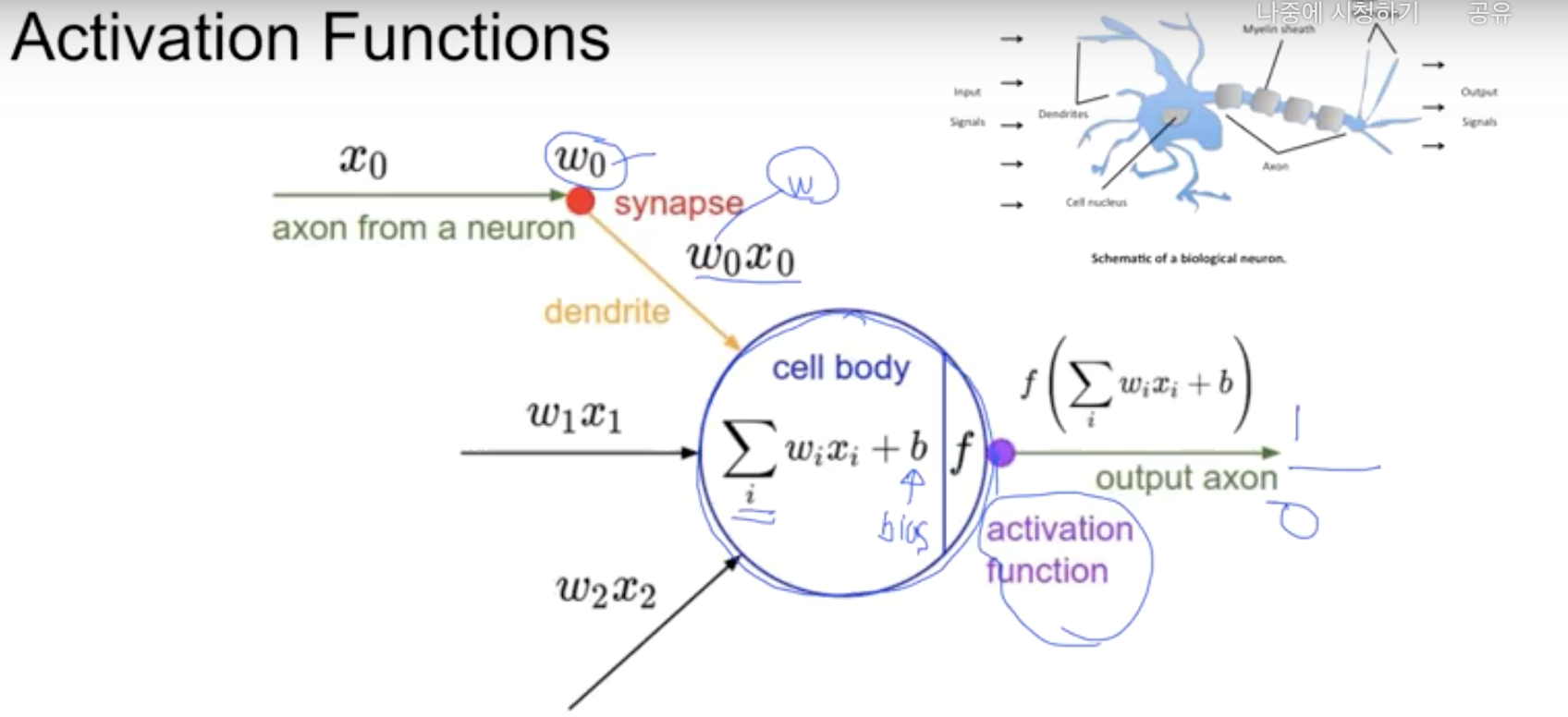
**섹션 8. 딥러닝의 기본 개념과 문제 그리고 해결**

**딥러닝의 기본 개념: 시작과 XOR 문제**



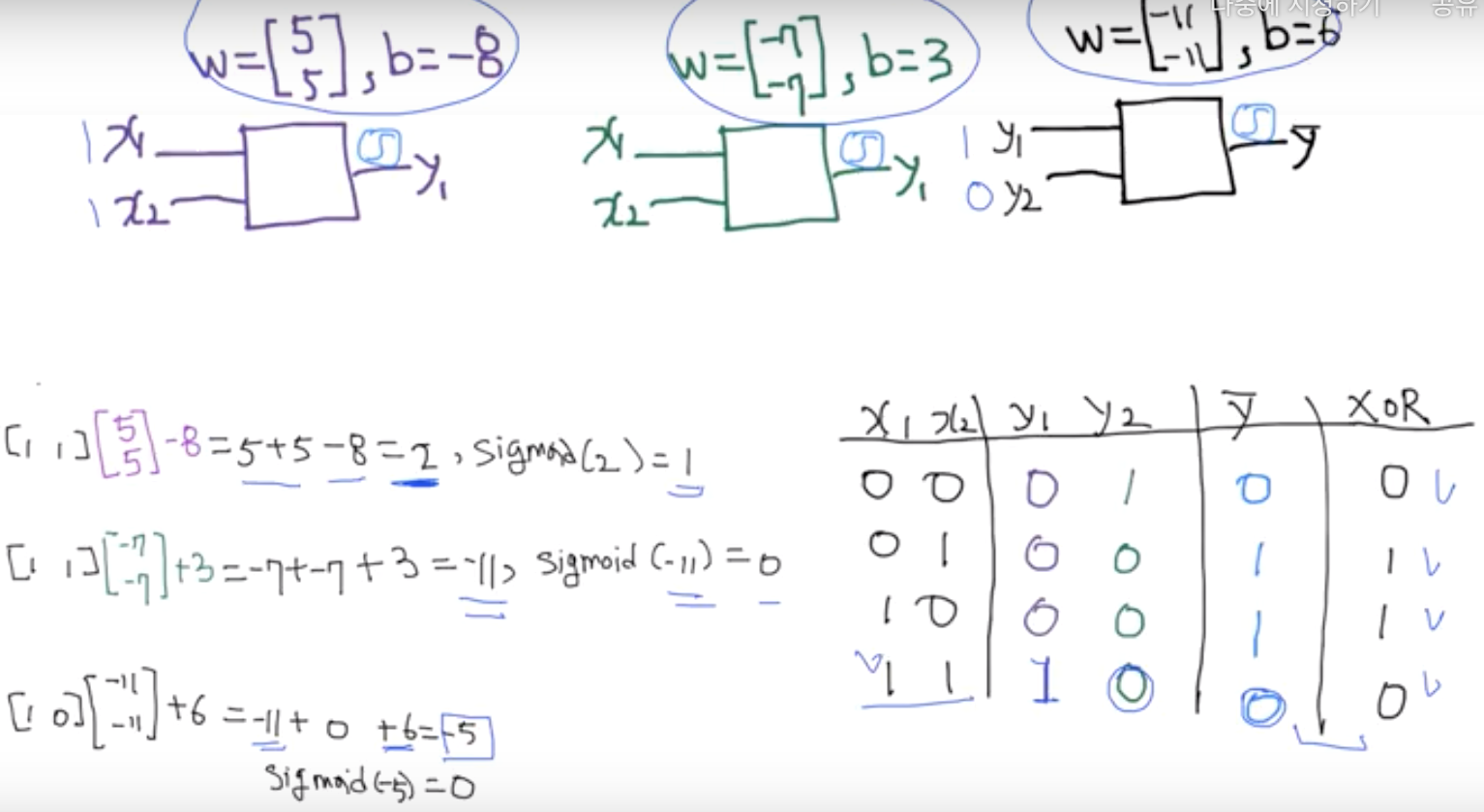
1. XOR 문제
   1. XOR은 AND, OR과는 달리 어떤 직선을 그어도 linear하게는 구분이 불가능함
      1. 하나의 unit으로는 불가능
   2. Perceptrons (1969, by Marvin Minsky of MIT AI Lab)
      1. XOR은 지금 상황에서는 풀 수 없다고 수학적으로 증명함
      2. MLP를 사용해야 하는데, 각 layer를 학습시킬 수가 없음…
         1. “No one on earth had found a viable way to train”
   3. Backpropagation (1974, 1982 by Paul Werbos, 1986 by Hinton)
      1. forward로 error를 구한 후, backward로 전달해나가면서 진행시키면 어떨까
      2. 복잡한 형태 예측 가능
   4. Convolutional Neural Networks
      1. 부분부분을 잘라서 보낸 다음에 나중에 합치는 방법
      2. ex. AlphaGo, 구글 자율주행 자동차
      3. 문자, 숫자 인식에 90% 이상의 성능을 보임

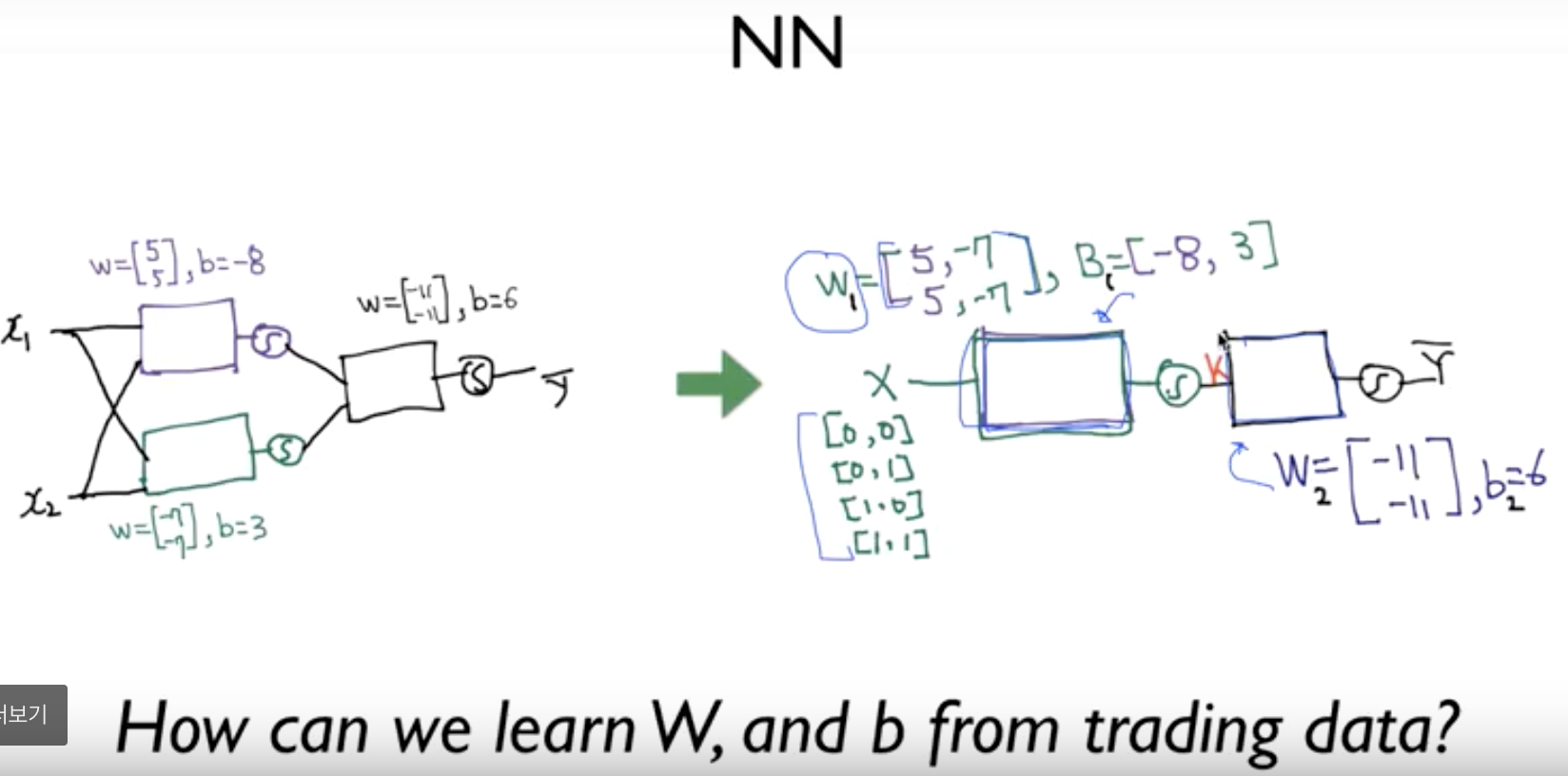
**딥러닝의 기본 개념2: Back-propagation과 2006/2007 ‘딥’의 출현**

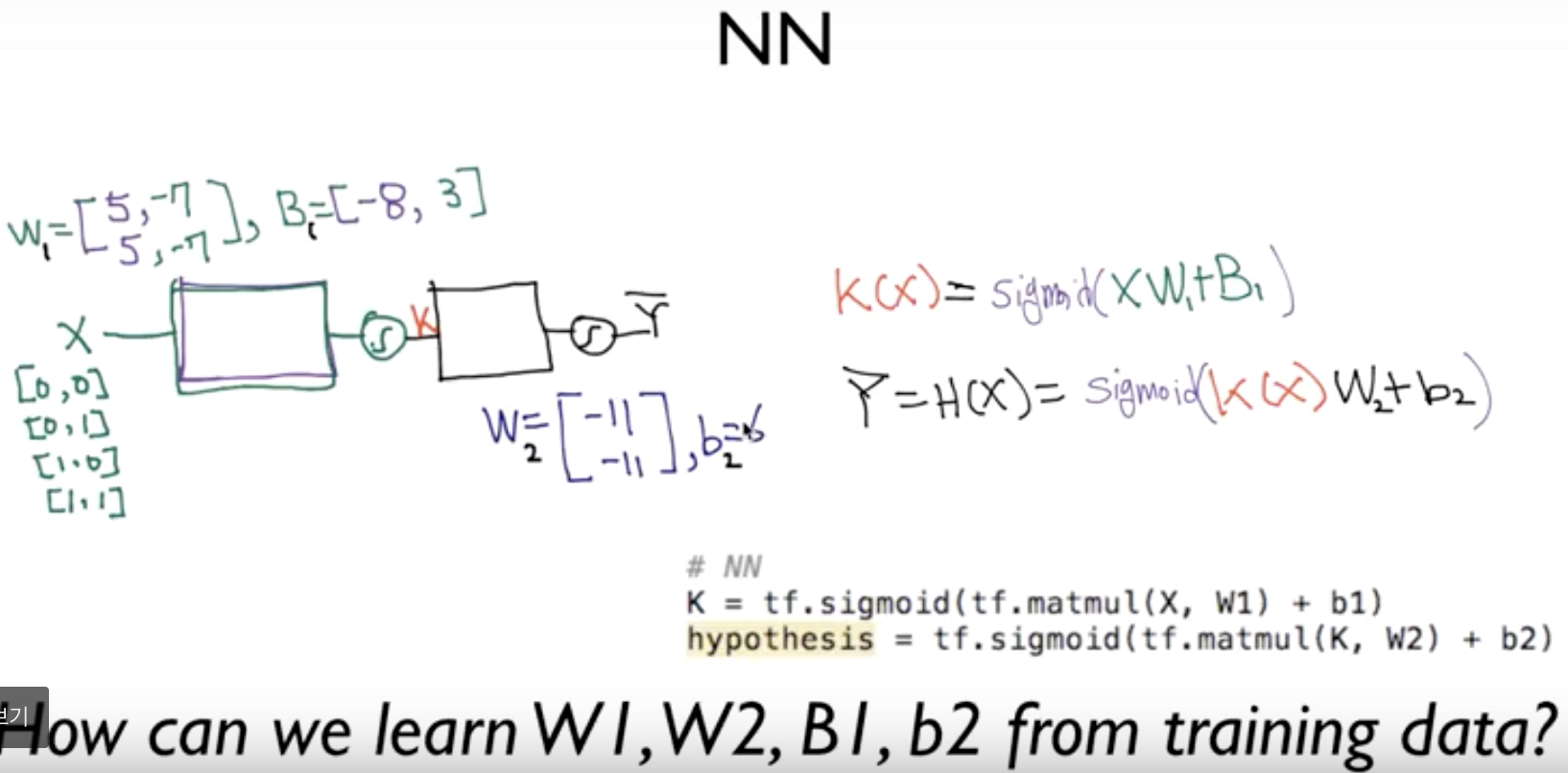
1. A BIG problem
   1. Backpropagation이 몇 개의 layer에서는 잘 동작하지만, 10여개의 layer에서는 error가 마지막 layer까지는 거의 전달되지 않아 학습 성능이 떨어짐
      1. Neural Network의 침체기
   2. Canadian Institute for Advanced Research (CIFAR)
      1. encourages basic research without direct application
      2. 아무도 이 일을 하지 않고 있을 때, worst possible time일 때 연구를 계속할 수 있도록 격려해줌
      3. Hinton과 Bengio의 논문 발표
         1. 초기값을 잘 선택한다면 layer가 많아도 neural network는 학습될 수 있다.
         2. 깊게 신경망을 구축하면 복잡한 문제를 풀 수 있다.
         3. **=> Rebranding to Deep Nets, Deep Learning**
2. ImageNet Classification
   1. 이미지를 주고 무엇인지를 인식하는 Computer Vision의 중요한 문제
   2. 매년 1-2%씩 발전하는 중이었음. 90% 이상은 되어야 사용할 만 함
   3. 그러다 2012년, error가 26.2%에서 15.3%로 훅 떨어짐!
   4. 2015년의 경우 3%대로 떨어짐 (딥러닝 기반 시스템)
      1. Human(스탠포드 학생): 5%
      2. 그림 구분뿐만 아니라 설명도 가능하게 됨
3. Deep API Learning
   1. 자연어를 주면(명령), 해당 명령을 수행하기 위해 필요한 API를 순서대로 알려줌
   2. 딥러닝 사용 전에는 약 25%의 정확도였으나, 딥러닝을 통해 이 시스템을 만들었더니 정확도가 65%까지 상승했음
4. 딥러닝의 활용
   1. 바이두 Deep Speech
   2. 테트리스 게임을 사람보다 더 잘 함
   3. AlphaGo
5. 이전에는 왜 잘 안 되었을까?
   1. 다음 학습에서 자세히 다룸
6. 중요성
   1. 소리를 듣고 자동으로 생성되는 Youtube 자막
   2. Facebook 성능 향상: 관심 가질 만한 피드를 학습을 통해 보여줌
   3. 구글 키워드 검색: 좋아할 만한 문서를 학습을 통해 예측함
   4. 넷플릭스, 아마존: 추천 시스템을 통해 크게 성장함
   5. 손님들이 어떤 물건을 잘 사는지(앞쪽에 어떤 물건을 진열하면 좋을지)
7. 왜 지금 해야 할까
   1. 지금 시작해도 세계적인 전문가가 되기에는 늦지 않았다.
   2. 그렇게 복잡하지 않음 (수학적으로도)
   3. 실용적 - 현재의 딥러닝은 90% 이상의 정확도
      1. many ready-to-use tools such as TensorFlow
      2. many easy/simple programming languages such as Python

**섹션 9. Neural Network 1: XOR 문제와 학습방법, Backpropagation**

**XOR 문제 딥러닝으로 풀기**

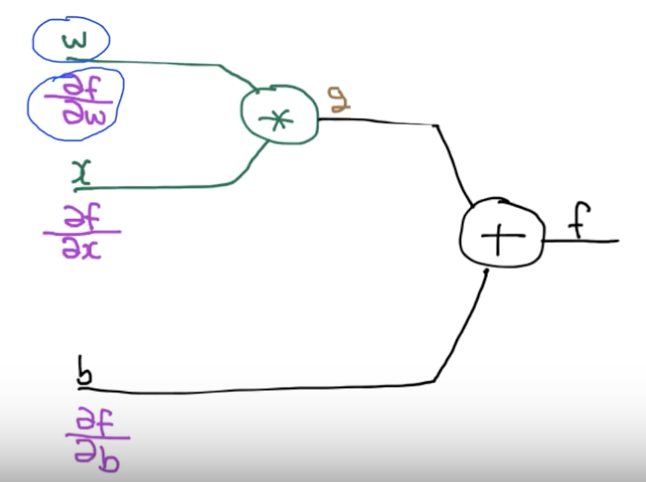
****





**딥넷트웍 학습시키기 (backpropagation)**

1. Gradient Descent Algorithm
   1. y hat으로 cost 함수를 정의
   2. 초기 w값에서의 기울기를 구하고 계속 내려가다 보면 global minimum에 도착할 수 있다. (cost 생성 가능)
      1. 기울기 -> 미분값 필요
      2. 각 cell에서의 x가 y hat에 미치는 영향을 알아야 weight을 조절할 수 있는데, 계산하기 어렵다.
         1. Backpropagation 알고리즘으로 해결
2. Backpropagation (chain rule)
   1. f = wx + b, g = wx, f = g+b
   2. w, x, b가 f에 미치는 영향을 알고 싶음
      1. 미분값을 알아야 함
      2. chain rule을 이용해 local derivative를 곱해 나감



**Neural Network 2:**

**ReLU and 초기값 정하기 (2006/2007 breakthrough)**

**XSigmoid보다 ReLU가 더 좋아**

1. 여러 개의 layer를 사용해 딥러닝을 했지만 cost가 높게 나오고 학습이 잘 되지 않는 경우
   1. vanishing gradient 문제
2. Vanishing gradient (NN winter 2:1986-2006)
   1. 최종 layer에서 가장 첫 layer로 갈수록(backpropagation) 경사도가 사라진다. 예측이 되지 않는다.
      1. sigmoid => 0~1의 값
      2. 0.01\*0.01\*... ~ 0
   2. non-linearity 즉 sigmoid를 잘못 사용했기 때문 (1보다 작은 값이라서)
      1. by Geoffrey Hinton
   3. Sigmoid 대신 ReLU라는 activation function을 사용하자.
3. Rectified Linear Unit (ReLU)
   1. **max(0, X)**
   2. L1 = tf.**nn.relu**(tf.matmul(X, W1) + b1)
      1. **마지막 단에서만 sigmoid를 사용**해 0~1 사이의 값을 출력
4. Leaky ReLU
   1. **max(0.1\*x, x)**
5. 기타
   1. ELU
      1. f(x) = x if x>0, alpha\*(exp(x) - 1) if x<=0
   2. tanh
      1. sigmoid를 y축 방향으로 이동한 것과 유사한 모습 (-1~1)

**Weight 초기화 잘해보자**

1. weight의 초기값에 따라서도 cost 값이 달라진다.
   1. 초기화를 잘(현명하게) 해야 한다.
   2. W =0일 경우 gradient가 아예 사라져버림
      1. Not all 0’s
2. **Restricted Boatman Machine (RBM)**
   1. **RBM을 사용해 초기화시킨 네트워크 = Deep Belief Net**
   2. forward(x) (encode)
   3. backward =>x bar (decode)
   4. 두 레이어에 대해서만 각각 진행 (전체 네트워크 내에서)
   5. **x와 x bar의 차가 최소가 되도록 weight을 조절!**
      1. fine tuning
3. Good news
   1. weight 초기화에 RBM을 쓰지 않아도 됨
   2. simple method도 OK
      1. Xavier initialization(2010)
         1. 한 node의 입력/출력 갯수에 비례해서 초기값을 선택
            1. input의 수: fan\_in
            2. output의 수: fan\_out
         2. W = np.random.randn(fan\_in, fan\_out) / np.sqrt(fan\_in)
      2. He initialization
         1. W = np.random.randn(fan\_in, fan\_out) / np.sqrt(fan\_in / 2)

**Dropout과 앙상블**

1. Overfitting
   1. very high accuracy on the training dataset (0.99)
   2. **poor accuracy on the test data set (0.85)**
   3. layer가 많아질수록 error가 떨어져야 하는데, test dataset에서는 어느 순간부터 error가 높아짐
2. Regularization
   1. Let’s not have too big numbers(lambda) in the weight
      1. cost + lambda \* sum of w^2
         1. regularization strength (lambda)
      2. l2reg = 0.001 \* tf.reduce\_sum(tf.square(W))
3. Dropout (Srivastava et al. 2014)
   1. “randomly set some neurons to zero in the forward pass”
      1. NN을 학습할 때에는 일부 네트웍을 끊어버림(몇 개의 node를 죽여버림)
   2. neuron을 random하게 쉬게 해서 훈련시킴
   3. 마지막에는 총 동원해서 예측시킴
   4. 학습 도중에만 dropout 시키고, 실전에서는 모두 사용
   5. TRAIN:
      1. sess.run(optimizer, feed\_dict={X: batch\_xs, Y: batch\_ys, **dropout\_rate: 0.7**})
      2. 70%만 참여해
      3. 보통 0.5~0.7
      4. 실전에서는(EVALUATION) 반드시 1로 해야 함
   6. EVALUATION:
      1. print(“Accuracy:”, accuracy.eval({X: mnist.test.images, Y:mnist.test.labels, **dropout\_rate: 1**})
4. Adam
   1. MNIST Multilayer NN + dropout 시 Adam을 이용하면 training cost가 굉장히 빨리 떨어짐
   2. optimizer = tf.train.**AdamOptimizer**(learning\_rate=learning\_rate).minimize(cost)
5. Ensemble
   1. 독립적인 NN을 여러 개 준비해 data set을 학습시킨 후 combine
      1. 동일하거나 서로 다른 training set을 이용해 학습
      2. 독립적인 여러 명의 전문가에게 물어보는 것과 같음
   2. 2% ~ 4-5%대까지 성능 향상 가능

**레고처럼 넷트웍 모듈을 마음껏 쌓아 보자**

1. Feedforward neural network
   1. n단
2. Fast forward
   1. n단 앞으로 추가
3. Split & merge
4. Recurrent network(RNN)
   1. 앞으로뿐만 아니라 옆으로도 나감
5. 어떤 형태든지 네트웍을 조합해 문제를 풀 수 있다.
   1. ‘The only limit is your imagination’

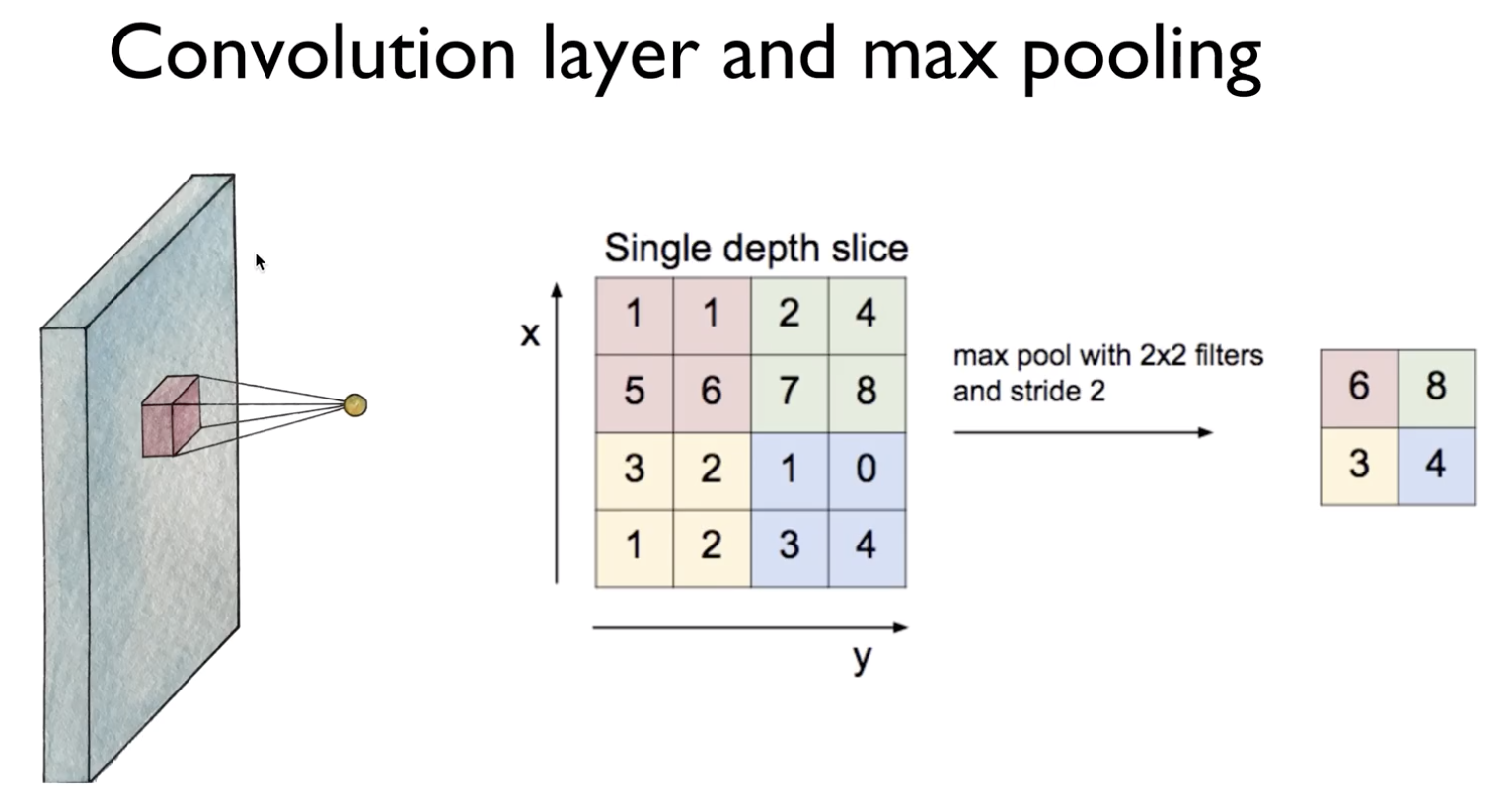
**섹션 11. Convolutional Neural Networks**

**ConvNet의 Conv 레이어 만들기**

1. 이미지 전체 대신 이미지의 일부분부터 처리
   1. filter를 사용해 filter 크기만큼 일부분을 읽어들임
      1. image (width \* height \* depth(rgb=>3))
   2. filter 내에서 하나의 number를 추출함
      1. Wx + b
      2. ReLU(Wx + b)
   3. 이미지 전체에서 filter별로 각각의 점을 추출
      1. stride=1 => filter를 한 칸씩 움직임
   4. N\*N image, F\*F filter
      1. output size: (N-F) / stride + 1
      2. stride 1=> output size=5
      3. stride 2=> output size=3
      4. …
      5. image가 작아질수록 정보를 잃어버린다.
      6. => 실제로 사용할 때에는 padding을 사용
         1. **7\*7** => 9\*9 (pad with 1 pixel border, 3\*3 filter, stride 1)
            1. output size = **7\*7 (input size와 동일)**
         2. 그림이 급격하게 작아지는 것을 방지
         3. 모서리의 위치를 네트웍에 알려줌
2. activation maps (output size, output size, filter 갯수)
3. activation map에 또다시 filter를 이용해 convolution layer를 적용
4. weight variable
   1. random하게 초기화
   2. 우리가 가진 데이터로 학습

**ConvNet Max pooling 과 Full Network**

1. Pooling layer (sampling)
   1. image -> conv layer -> resize(sampling)
2. Max pooling



**ConvNet의 활용 예**