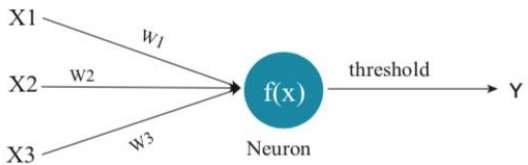
Neural Network

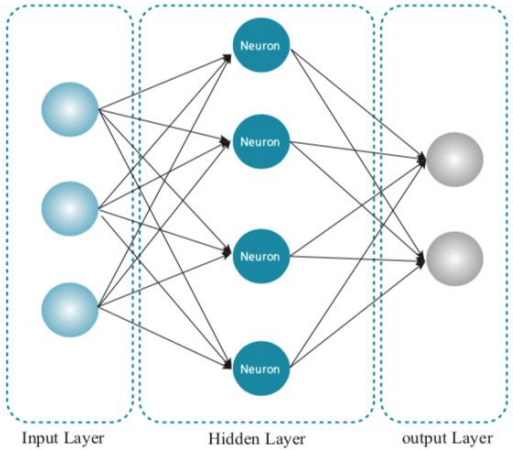
Deep Learning

- Deep Learning 이란
  - 여러 층을 가진 인공 신경망(Artificial Neural Network)을 사용하여 머신러닝 학습을 수행
  - 딥러닝은 기계가 자동으로 학습하려는 데이터에서 특징을 추출하여 학습

- 인공신경망(Artificial Neural Network)
  - 인공 신경망은 인간의 신경세포 뉴런(Neuron)과 같은 서로 연결된 뉴런은 서로의 입력신호와 출력 신호를 이용하여 동작함
  - 뉴런과 신경망 연결 구조



뉴런 동작



신경망 구성

Deep Learning

- 뉴런 동작 과정
  - 다수의 입력 신호 X와 W를 이용하여 f(x)에서 학습
  - 학습된 출력 결과는 threshold(Decision boundary, 결정 경계, 임계값)를 기준으로 한 활성화 함수(Activation Function)로 결과를 판단하여 결괏값 출력

- 인공신경망 동작 과정
  - 구성 : Input Layer(입력층), Hidden Layer(은닉층), Output Layer(출력층)
  - 동작 과정
    (1) Input Layer를 통하여 학습데이터를 입력 받음
    (2) 여러 단계의Hidden Layer를 지나면서 계산(학습)된 결과를 Output Layer로 전달
    (3) Output Layer에서는 학습된 결과를 처리하여 최종 결과를 출력함
  - 심층 신경망(Deep Neural Network) : Input Layer, Hidden Layer, Output Layer을 3단계 이상 중첩한 구조

Backpropagation

- 일반적인 비용함수 최적화
  - Gradient descent 알고리즘을 이용하여 비용함수 미분을 통하여 오차가 최소가 되는 W(Weight), b(bias) 를 최적화함
  - 순전파(Forward propagation) 과정(Input->Hidden->Output Layer)을 통하여 미분값을 업데이트됨

- 신경망의 비용함수 최적화
  - 신경망에서는 미분값을 구하기 위하여 Backpropagation(역전파)알고리즘을 사용

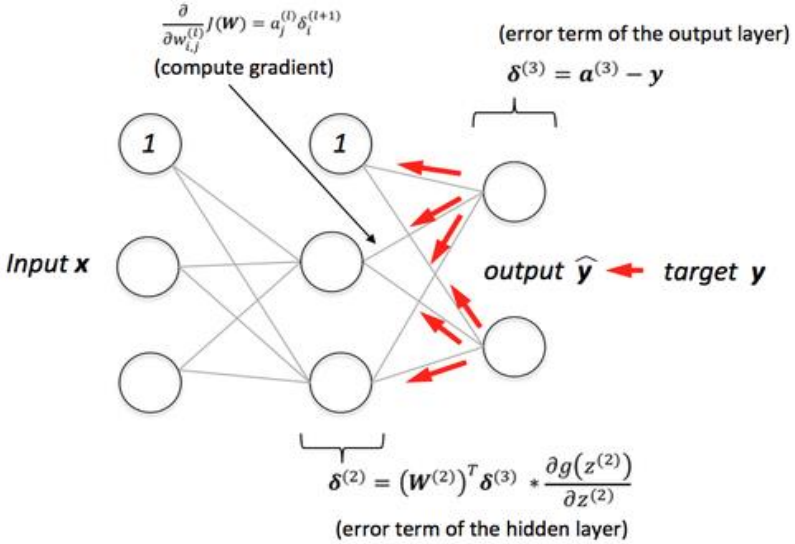
- Backpropagation 알고리즘 학습 과정
  - 신경망의 W(가중치)를 적당한 값으로 초기화
  - Input Layer에 학습데이터를 입력하여 순전파(Foward propagation) 과정을 통하여 에서 비용함수의 미분값 연산 수행
  - Output Layer의 출력한 예측값과 실제값의 오차를 계산
  - 계산된 오차를 신경망의 각각의 뉴런들에 오차를 역전파(Backpropagate)하여 에러값을 이전 Layer로 전달
  - 전달되 오차는 뉴런들의 W로 사용되며, 오차가 최소가 되는 W, b 를 최적화 함

❏ **Forward propagation**: Input Layer로 입력된 학습데이터로부터 예측값을 계산하고, 각 Ouput Layer 뉴런에서의 오차를 계산.
Input -> Hidden -> Output 으로 정보가 흘러가기 때문에 'Forward' propagation이라 함

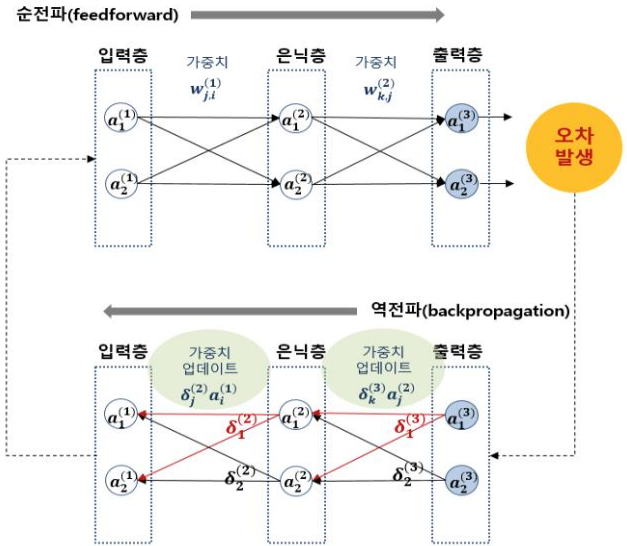
❏ **Backpropagation**: Output Layer 뉴런에서 계산된 오차를 각 edge들의 weight를 사용해 바로 이전 Layer의 뉴런들이 얼마나 오차에 영향을 미쳤는지 계산.
Output -> Hidden Layer 으로 정보가 흘러가기 때문에 'Back' propagation이라 함

Backpropagation

- Backpropagation 알고리즘을 이용한 모델 학습 과정
  - 순전파 -> 역전파 -> 가중치업데이트 -> 순전파 -> 역전파 -> 가중치 업데이트 ->… 과정을 반복하여 예측값과 결괏값의 오차가 최소가 되는 W, b를 찾음



출처 : https://sebastianraschka.com/faq/docs/visual-backpropagation.html

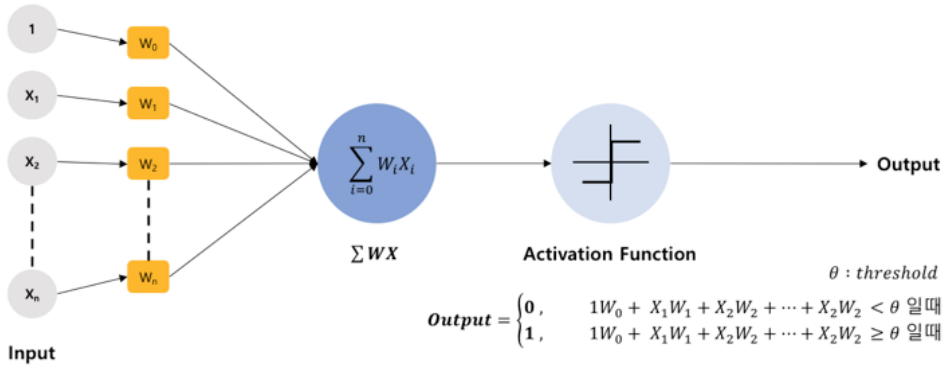


출처 : https://m.blog.naver.com/samsjang/221033626685?view=img_75

  - 역전파 데모 참고 자료 : https://google-developers.appspot.com/machine-learning/crash-course/backprop-scroll/?hl=ko

Perceptron

- 퍼셉트론(Perceptron)
  - 가장 간단한 인공 신경망 구조
  - 다수의 신호(Input)를 입력받아서 하나의 신호(Output)를 출력
  - 퍼셉트론 동작 순서
    - 각각의 입력 신호에 부여된 W(Weight)와 계산
    - 계산 결과의 총합이 활성화 함수(Activation Function)로 입력
    - 활성화 함수에서는 정해진 임계값(threshold)을 넘었을때 1을 출력 넘지 못한 경우 0 혹은 -1 을 출력
  - W값이 크면 해당 신호는 중요한 신호라고 판단하게 됨
  - 일반적으로 퍼셉트론에서 사용되는 활성화 함수는 헤비사이드 계단함수(Heaviside Step Function)이 사용됨



Perceptron

- 퍼셉트론 결과값에서 임계값
  - 활성화 함수에서 사용하는 임계값(threshold)은 θ 로 표현
  - $1W_0 + X_1W_1 + X_2W_2 + \cdots + X_2W_2 < \theta$ 수식에서 θ 를 -b(bias, 편향)로 치환하여 수식 을 변경

$$\boldsymbol{Output} = \begin{cases} 0, & 1W_0 + X_1W_1 + X_2W_2 + \cdots + X_2W_2 < \theta \\ 1, & 1W_0 + X_1W_1 + X_2W_2 + \cdots + X_2W_2 \geq \theta \end{cases} \Rightarrow \boldsymbol{Output} = \begin{cases} 0, & \mathrm{b} + 1W_0 + X_1W_1 + X_2W_2 + \cdots + X_2W_2 < 0 \\ 1, & \mathrm{b} + 1W_0 + X_1W_1 + X_2W_2 + \cdots + X_2W_2 \geq 0 \end{cases}$$

  - 편향(bias)는 학습 데이터(입력신호)와 가중치(Weight)의 계산에 의한 값이 넘어야 할 값
  - 편향보다 높으면 1 혹은 0으로 분류되는 기준이 높아지기 때문에 분류할때 엄격하게 분류하게됨
  - 편향값이 높을 수록 학습 모델은 간단해지는 경향을 보이고 Underfitting(과소적합)이 될 수 있음
  - 편향값이 낮을 수록 학습 모델은 복잡해지는 경향을 보이고 Overfitting(과적합)이 될 수 있음

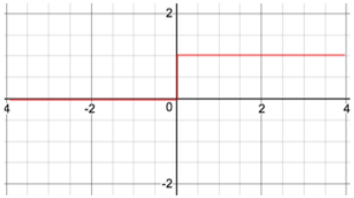
- W 역할 : 입력 신호가 결과 출력에 주는 영향을 조절
- b 역할 : 얼마나 쉽게 활성화(결과를 1로 출력)되는지를 조절
- 다층 퍼셉트론(Multi Layer Perceptron, MLP - 다수의 퍼셉트론 사용하는 신경망)을 활용하여 어려운 문제 혹은 비선형적 문제를 해결 할 수 있음

Activation Function

- Activation Function(활성화 함수)
  - threshold(임계값)을 이용하여 출력값을 결정하는 함수
  - 출력값에 따라서 다음 단계(뉴런) 의 입력값의 상태를 결정하게 됨

- 종류
  - Step Function
    - 가장 기본이 되는 활성화 함수로 계단 형태를 가지고 있음
    - 0을 기준으로 0 혹은 1을 출력

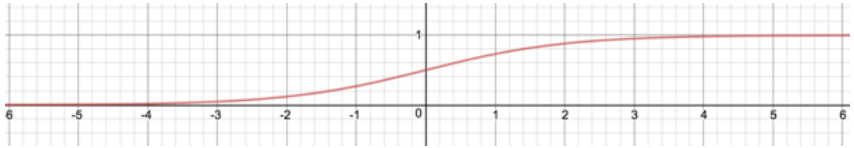
$$Output = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}$$



Activation Function

- 종류
  - Sigmoid Function
    - 0과 1사이의 값만 가질수 있도록 하는 비선형 함수
    - Step Function은 0과 1의 출력값만 가졌지만 Sigmoid Function은 0~1 사이의 연속적인 출력값을 가짐

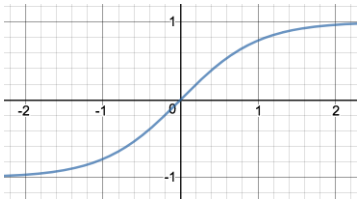
$$P = \frac{1}{1 + e^{-X}}$$



    - 신경망 초기에는 많이 사용되었지만 Gradient Vanishing 현상이 발생하여 최적화가 안되는 현상이 발생하여 최근에는 많이 사용하지 않음

  - Hyperbolic Tangent Function, tanh
    - 함수의 중심값을 0으로 옮겨 출력값의 범위는 -1~1 사이의 연속적인 출력값가지는 비선형 함수
    - Sigmoid Function 보다 최적화 가 빠름
    - Gradient Vanishing 현상이 발생함

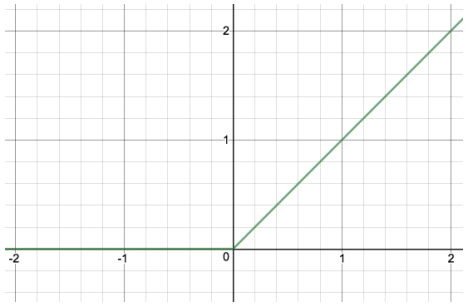
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Activation Function

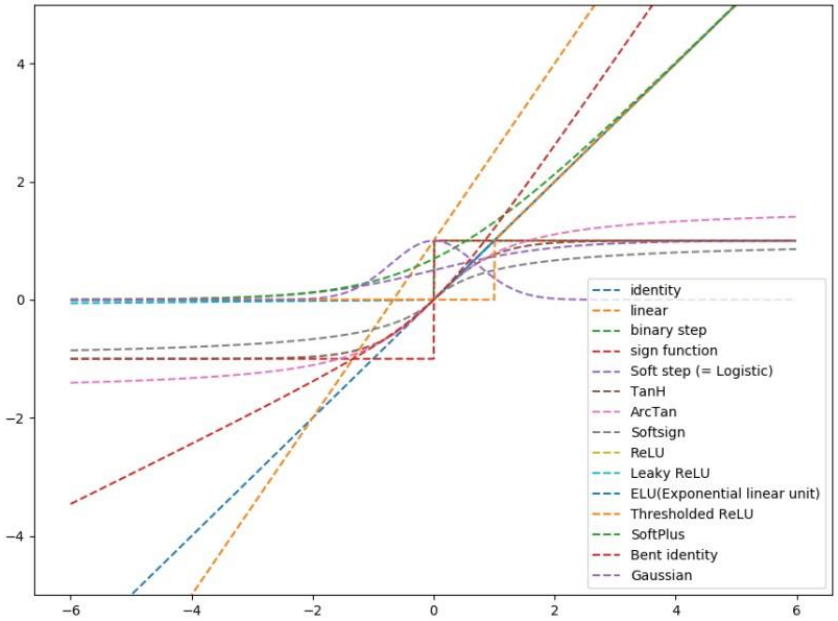
- 종류
  - ReLU(Rectified Linear Unit) Function
    - 최근 많이 사용되는 활성화 함수로 x가 0보다 크면 기울기가 1인 직선을 가짐
    - Sigmoid, tanh Function보다 학습이 빠르며 구현이 쉬움
    - x 가 0보다 작은 값들에 대해서는 미분시 기울기가 0이기 때문에 뉴런이 활성화가 되지 않음

$$f(x) = \max(0, x)$$



Activation Function

- 종류
  - 이외의 Activation Function 그래프



출처 : https://mblogthumb-phinf.pstatic.net/MjAxNzA2MDNfMTQ2/MDAxNDk2NDU0NjE5OTY1.KDNgrWWc2BIWJzitH-7kd6HkA_7tR-uBhSA1SBNhBdgg.-G6q8LTex-T7CvoRCSkuCfULFEFoGSjHa6TxkA7Qm58g.JPEG.wideeyed/%25EC%25A0%2584%25EC%25B2%25B4%25EA%25B7%25B8%25EB%259E%2598%25ED%2594%2584.jpg?type=w800

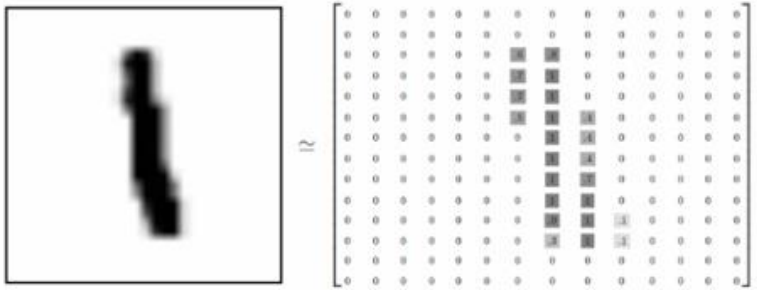
MNIST Neural Network

MNIST Neural Network 개요

- MNIST
  - MNIST(Modified National Institute of Standards and Technology database) 데이터세트
    - 손으로 쓴 숫자들로 이루어진 대형 데이터베이스
    - 다양한 화상 처리 시스템을 트레이닝 하기 위해 일반적으로 사용
    - 55,000개의 훈련데이터와 10,000개의 테스트 데이터 5,000개의 검증 데이터로 구성
    - 데이터 샘플 이미지

MNIST Neural Network 개요

- MNIST
  - 손글씨 이미지를 픽셀 데이터로 변환하여 학습에 사용할 수 있도록 함



  - 하나의 이미지는 28 x 28 픽셀로 구성되어 있으며 픽셀 데이터를 784(28*28)의 벡터로 변환하여 학습에 사용

MNIST Neural Network 실습

- **학습에 필요한 모듈 선언**

```
1  ######################################################################
2  # [학습에 필요한 모듈 선언]
3  ######################################################################
4  import tensorflow as tf
5  from tensorflow.examples.tutorials.mnist import input_data
6  import numpy as np
7  from matplotlib import pyplot as plt
8
```

  - tensorflow, numpy, matplotlib 라이브러리 사용

MNIST Neural Network 실습

- **환경 설정**
  - 학습률, 총 학습 횟수, 학습데이터 수 선언
  - W, b 변수 생성 타입을 설정
  - Neural Network 의 크기를 선언

- 배치 트레이닝
  - 학습데이터를 일정 사이즈로 나누고 미니 배치(mini batch)를 통해 Gradient descent를 수행하는 방법
  - 미니 배치를 이용하여 기울기를 계산하여 W, b를 업데이트 함
  - 전체데이터를 이용하여 W, b의 값을 한번에 업데이트 하는 방법보다 빠르게 W,b를 업데이트 할 수 있음
  - 배치 사이즈는 메모리가 허용하는 범위 내에서 최대한 크게 잡는것이 좋음
  - epoch은 전체 데이터 학습 하는 횟수

```
9  ##############################################################
10 # [환경설정]
11 ##############################################################
12 # 학습률
13 learningRate = 0.001
14
15 # 총 학습 횟수
16 totalEpochs = 20
17 # 학습데이터를 나누기 위한 값
18 # 학습데이터 총수 / batch_size = 한번의 epoch 쓰이는 데이터 수
19 batch_size = 200
20
21 # W, b 변수 생성 타입 (1 : random_normal, 2: truncated_normal, 3:  random_uniform)
22 randomVariableType = 1
23
24 # input Layer 크기
25 # 입력 데이터 크기 784 (손글씨 이미지는 28 * 28 픽셀로 총 784개)
26 inputDataSize = 28 * 28 # 입력 데이터 고정값(수정불가)
27
28 # hidden Layer 크기
29 hiddenLayer1Size = 1024
30 hiddenLayer2Size = 512
31 hiddenLayer3Size = 256
32
33 # output Layer 크기
34 # 출력값 크기 (Output Layer에서 출력되 데이터(0~9까지 숫자)
35 outputLayerSize = 128
36 outputDataSize = 10 # 출력값 크기 고정(수정불가)
37
```

MNIST Neural Network 실습

- **빌드단계 - Step1) 학습 데이터 준비**
  - Tensorflow 공식 Github에서 제공하는 input_data 모듈 사용하여 학습데이터를 다운로드
  - one_hot=True로 설정하여 실제값(라벨)을 One Hot Vector 형태로 다운로드 함

```
 1 2 3 4 5 6 7 8 9 0
[1 0 0 0 0 0 0 0 0 0]:1
[0 1 0 0 0 0 0 0 0 0]:2
[0 0 1 0 0 0 0 0 0 0]:3
[0 0 0 1 0 0 0 0 0 0]:4
[0 0 0 0 1 0 0 0 0 0]:5
[0 0 0 0 0 1 0 0 0 0]:6
[0 0 0 0 0 0 1 0 0 0]:7
[0 0 0 0 0 0 0 1 0 0]:8
[0 0 0 0 0 0 0 0 1 0]:9
[0 0 0 0 0 0 0 0 0 1]:0
```

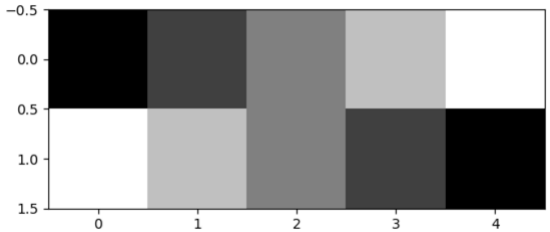
  - 다운 로드 한 MNIST 학습데이터 크기

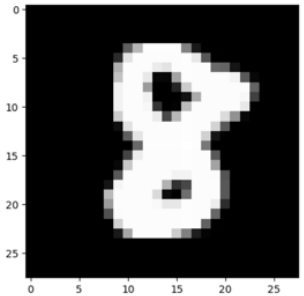
```
Train data num         : 55000
Train data shape       : (55000, 784)
Test data num          : 10000
Train data shape       : (10000, 784)
Validation data num    : 5000
Validation data shape  : (5000, 784)
```

```
38 ##############################################################
39 # [빌드단계]
40 # Step 1) 학습 데이터 준비
41 ##############################################################
42 # 공식 tensorflow github에서 제공하는 mnist dataset 다운로드
43 # 결과 데이터는 ont hot encoding을 적용
44 mnist = input_data.read_data_sets("./dataset", one_hot=True)
45
46 print("Train data num        : {}".format(mnist.train.num_examples))
47 print("Train data shape      : {}".format(mnist.train.images.shape))
48 print("Test data num         : {}".format(mnist.test.num_examples ))
49 print("Train data shape      : {}".format(mnist.test.images.shape))
50 print("Validation data num   : {}".format(mnist.validation.num_examples))
51 print("Validation data shape : {}".format(mnist.validation.images.shape))
52
53 # 손글씨 이미지 픽셀로 표현 방법
54 image = [[1, 2, 3, 4, 5],
55          [5, 4, 3, 2, 1]]
56 plt.imshow(image, cmap='gray')
57 plt.show()
58 # 손글씨 이미지 그래프로 출력
59 batch = mnist.train.next_batch(1)
60 plotData = batch[0]
61 plotData = plotData.reshape(28, 28)
62 plt.imshow(plotData, cmap='gray')
63 plt.show()
64
```

MNIST Neural Network 실습

- **빌드단계 - Step1) 학습 데이터 준비 : 데이터 읽기**
  - MNIST 데이터의 픽셀 데이터는 28 x 28(Width x Height) 총 784개의 픽셀로 구성됨
  - 픽셀 데이터는 색의 농도로 표현이 가능하며 MNIST 데이터를 matplotlib를 이용하여 이미지 표현 가능
    예) [[1, 2, 3, 4 ,5] , [5, 4, 3, 2, 1]] 데이터의 농도 표현

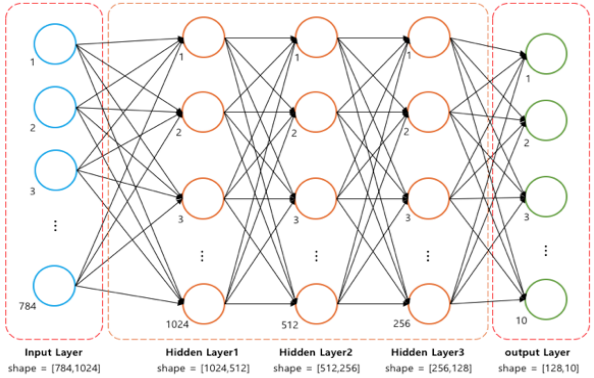




MNIST Neural Network 실습

- **빌드단계 - Step2) 모델 생성을 위한 변수 초기화**
  - 학습 데이터 입력공간 X, Y를 placeholder 로 선언
  - W와 b를 값을 저장할 변수를 Variable로 선언 하는데 환경설정에서 선언한 난수 타입으로 생성
  - Neural Network에 구성되는 뉴런의 W, b를 지정
  - 난수 생성 타입의 종류에 따라 W, b 를 선언함

- Network 구조




```
65  ############################################################
66  # [빌드단계]
67  # Step 2) 모델 생성을 위한 변수 초기화
68  ############################################################
69  # 학습데이터가 들어갈 플레이스 홀더 선언
70  X = tf.placeholder(tf.float32, [None, inputDataSize])
71  # 학습데이터가 들어갈 플레이스 홀더 선언
72  Y = tf.placeholder(tf.float32, [None, outputDataSize])
73
74  # 임의의 난수를 선언하여 W,b 변수의 초기값을 선언 및 Neural Network Layer 구성
75  if randomVariableType == 1:
76      # 1 : random_normal
77      # Input Layer
78      W_input = tf.Variable(tf.random_normal([inputDataSize, hiddenLayer1Size]),
79                            name='Weight_input')
80      b_input = tf.Variable(tf.random_normal([hiddenLayer1Size]),
81                            name='bias_input')
82      # Hidden Layer
83      # Layer1
84      W_hidden1 = tf.Variable(tf.random_normal([hiddenLayer1Size, hiddenLayer2Size]),
85                              name='Weight_hidden1')
86      b_hidden1 = tf.Variable(tf.random_normal([hiddenLayer2Size]),
87                              name='bias_hidden1')
88      # Layer2
89      W_hidden2 = tf.Variable(tf.random_normal([hiddenLayer2Size, hiddenLayer3Size]),
90                              name='Weight_hidden2')
91      b_hidden2 = tf.Variable(tf.random_normal([hiddenLayer3Size]),
92                              name='bias_hidden2')
93      # Layer3
94      W_hidden3 = tf.Variable(tf.random_normal([hiddenLayer3Size, outputLayerSize]),
95                              name='Weight_hidden3')
96      b_hidden3 = tf.Variable(tf.random_normal([outputLayerSize]),
97                              name='bias_hidden3')
98      # Output Layer
99      W_output = tf.Variable(tf.random_normal([outputLayerSize,outputDataSize]),
100                             name='Weight_output')
101     b_output = tf.Variable(tf.random_normal([outputDataSize]),
102                             name='bias_output')
```

MNIST Neural Network 실습

- **빌드단계 - Step3) 학습 모델 그래프 구성**
  - 학습 데이터의 특성을 대표하는 **가설 수식 작성**
  - 가설 수식은 Network Layer마다 서로 다르게 구성할 수 도 있음
  - 가설 수식에 학습데이터 x 의 값을 입력한 결괏값(예측값)과 실제값의 오차를 계산하는 **비용함수(오차함수, 손실 함수) 선언**
  - Softmax를 사용하여 10개의 손글씨 이미지를 분류하여 예측값이 출력 되게 함
  - 비용함수의 값이 최소가 될 수 있도록 W, b의 최적값을 찾는 **최적화 함수 선언**
  - 최적화 함수는 Gradient descent 알고리즘 대신 Adam Optimizer를 사용

```
163 ############################################################
164 # [빌드단계]
165 # Step 3) 학습 모델 그래프 구성
166 ############################################################
167 # 3-1) 학습데이터를 대표 하는 가설 그래프 선언
168 # hypothesis - Input Layer
169 Layer_input_hypothesis = tf.nn.relu(tf.matmul(X, W_input)+b_input)
170 # hypothesis - Hidden Layer
171 Layer_hidden1_hypothesis = tf.nn.relu(tf.matmul(Layer_input_hypothesis, W_hidden1)+b_hidden1)
172 Layer_hidden2_hypothesis = tf.nn.relu(tf.matmul(Layer_hidden1_hypothesis, W_hidden2)+b_hidden2)
173 Layer_hidden3_hypothesis = tf.nn.relu(tf.matmul(Layer_hidden2_hypothesis, W_hidden3)+b_hidden3)
174 # hypothesis - Output Layer
175 Layer_output_hypothesis_logit = tf.matmul(Layer_hidden3_hypothesis, W_output)+b_output
176
177 # 3-2) 비용함수(오차함수,손실함수) 선언
178 costFunction = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
179                                logits=Layer_output_hypothesis_logit,
180                                labels=Y))
181
182 # 3-3) 비용함수의 값이 최소가 되도록 하는 최적화함수 선언
183 optimizer = tf.train.AdamOptimizer(learning_rate=learningRate)
184 train = optimizer.minimize(costFunction)
185
186
```

MNIST Neural Network 실습

- **실행단계 - 학습 모델 그래프 실행**

```
187  ###############################################################
188  # [실행단계]
189  # 학습 모델 그래프를 실행
190  ###############################################################
191  # 실행을 위한 세션 선언
192  sess = tf.Session()
193  # 최적화 과정을 통하여 구해질 변수 W,b 초기화
194  sess.run(tf.global_variables_initializer())
195
196  # 예측값, 정확도 수식 선언
197  predicted = tf.equal(tf.argmax(Layer_output_hypothesis_logit, 1), tf.argmax(Y, 1))
198  accuracy = tf.reduce_mean(tf.cast(predicted, tf.float32))
199
200  # 학습 정확도를 저장할 리스트 선언
201  train_accuracy = list()
202
```

  - Session 변수(sess)를 선언
  - 최적화 과정에서 계산되는 변수(W, b)의 초기화
  - 예측값, 정확도를 구하기 위한 수식 선언(One Hot Encoding이용)
  - 모델 학습 결과 확인을 위한 리스트 선언

MNIST Neural Network 실습

- **실행단계 - 학습 모델 그래프 실행**
  - 배치 트레이닝 방법으로 모델 학습
  - totalEpochs 만큼 학습을 하고 epoch마다 batch_size 만큼의 학습데이터를 학습
  - 학습을 하면서 중간 결과를 저장하고 출력함
  - 정확도 결과 확인 그래프 출력

- 학습 조건
  - 훈련 데이터수 : 55,000개
  - 최적화 함수 : Adams Optimizer
  - 학습률 : 0.001
  - totalEpochs : 20회
  - totalBatch : 275회
  - batch_size : 200개 데이터
  - Neural Network Layer 구성은 환경설정에서 선언한 크기로 구성함

```
203 print("-------------------------------------------------------------------")
204 print("Train(Optimization) Start ")
205 for epoch in range(totalEpochs):
206     average_costFunction = 0
207     # 전체 batch 사이즈 구하기 (55000 / 200 = 275)
208     totalBatch = int(mnist.train.num_examples / batch_size)
209 
210     for step in range(totalBatch):
211         batchX, batchY = mnist.train.next_batch(batch_size)
212         cost_val, acc_val, _ = sess.run([costFunction, accuracy, train],
213                                         feed_dict={X: batchX, Y: batchY})
214         train_accuracy.append(acc_val)
215         average_costFunction = cost_val / totalBatch
216 
217     print("epoch : {}, cost = {}".format(epoch, average_costFunction))
218 
219 # 정확도 결과 확인 그래프
220 plt.plot(range(len(train_accuracy)),
221          train_accuracy,
222          linewidth=2,
223          label='Training')
224 plt.legend()
225 plt.title("Accuracy Result")
226 plt.show()
227 
228 print("Train Finished")
```

MNIST Neural Network 실습

- **실행단계 - 학습 모델 그래프 실행**

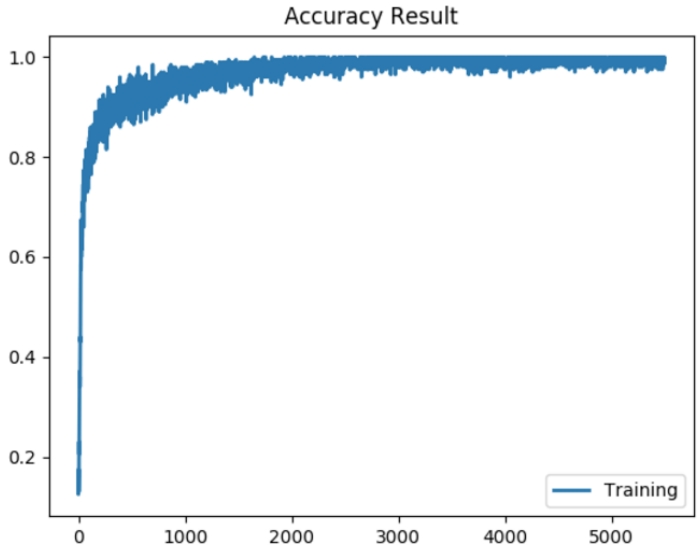
```
229  print("------------------------------------------------------------------------------------")
230  print("[Test Result]")
231  # 최적화가 끝난 학습 모델 테스트
232  h_val, p_val, a_val = sess.run([Layer_output_hypothesis_logit, predicted, accuracy],
233                                 feed_dict={X: mnist.test.images,
234                                            Y: mnist.test.labels})
235  print("\nHypothesis : {} \nPrediction : {} \nAccuracy : {}".format(h_val,
236                                                                    p_val,
237                                                                    a_val))
238
```

  - 테스트 데이터를 이용하여 학습을 완료한 모델의 결과를 확인함
  - 테스트 결과를 matplotlib를 이용하여 결과를 시각화 함(시각화 코드는 생략)

MNIST Neural Network 실습

- **결과 확인**
  - 20회 학습 결과 비용함수(오차)는 1번째 학습시 61에서 시작 하였으며 마지막 20번째 학습시 2로 줄어듬
  - 학습이 진행하면서 정확도는 그래프는 90이상의 정확도를 보임

```
Train(Optimization) Start
epoch : 0, cost = 61.05854403409091
epoch : 1, cost = 12.440392400568182
epoch : 2, cost = 7.903037109375
epoch : 3, cost = 3.335279873934659
epoch : 4, cost = 8.026814630681818
epoch : 5, cost = 2.8777567915482956
epoch : 6, cost = 3.7888805042613636
epoch : 7, cost = 3.588871848366477
epoch : 8, cost = 0.24910000887784092
epoch : 9, cost = 1.380424471768466
epoch : 10, cost = 0.5915650523792614
epoch : 11, cost = 0.0
epoch : 12, cost = 0.0
epoch : 13, cost = 0.43730654629794036
epoch : 14, cost = 1.5675537109375
epoch : 15, cost = 0.9786494584517046
epoch : 16, cost = 0.7234701815518466
epoch : 17, cost = 0.016692045385187322
epoch : 18, cost = 1.3333852317116477
epoch : 19, cost = 2.0624465110085226
Train Finished
```



MNIST Neural Network 실습

- **결과 확인**
  - 가설 수식의 결과값과 예측결과, 정확도를 출력함
  - 테스트 데이터를 이용하여 학습을 완료한 모델의 결과 95% 정도의 정확도를 보임
  - 예측 결과를 matplotlib를 이용하여 출력함
    - 이미지의 라벨 앞자리는 예측값
    - 이미지의 라벨 뒷자리는 실제값

```
[Test Result]

Hypothesis : [[  67801.62  -197245.86    72231.18  ...  469097.38   -58134.07
   246778.2  ]
 [ 102994.56     9881.387  267938.75  ... -157908.66   -59262.727
    17939.893]
 [ -51131.695  166566.02   -21273.182 ...   56397.824   43496.273
    24324.928]
 ...
 [ -80804.1     12043.096   50751.56  ...  216402.33   245847.47
   216147.55 ]
 [ 119363.234   72998.87    22246.812 ...   35118.47   235694.78
     5454.885]
 [  26286.031   18148.828  164208.    ...  -82760.37    31768.498
   -78076.73 ]]
Prediction : [ True  True  True ...  True  True  True]
Accuracy : 0.9503999948501587
```