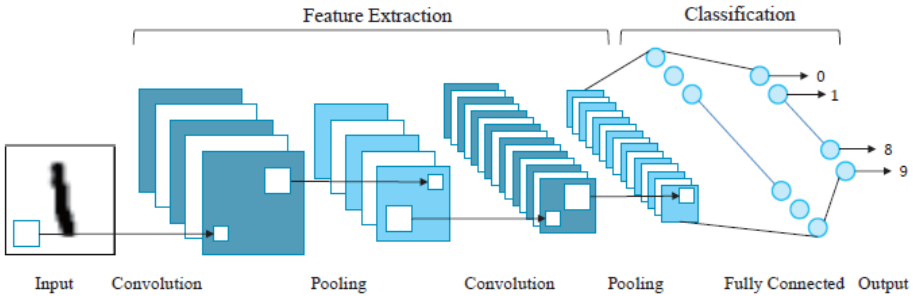
합성곱 신경망

Convolution Neural Network

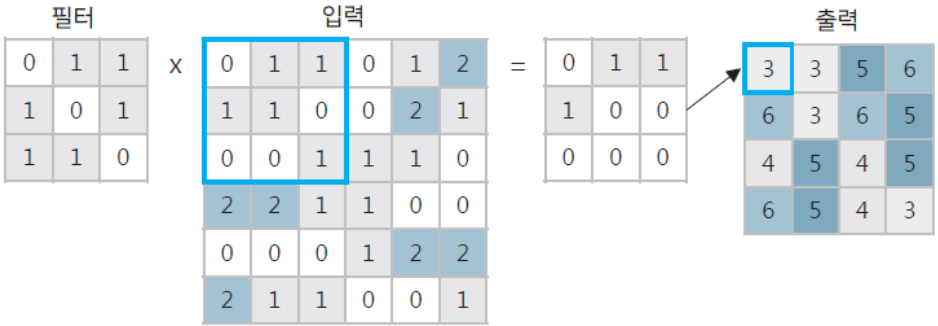
합성곱 신경망(CNN) 구현

- 합성곱 신경망 이란?
  - 1998년 Yann Lecun이 처음 제안한 알고리즘
  - 페이스북의 자동 사진 태그, Google과 네이버의 이미지 검색, 아마존의 제품 추천, 카카오의 형태소 분석기 등
- 합성곱 신경망 구조
  - 합성곱 계층, 풀링 계층, 완전 결합 계층으로 구성



합성곱 신경망(CNN) 구현

- 합성곱 계층
  - 특징을 추출하기 위한 필터(filter) / 커널(kernel)
  - 이미지의 행렬을 합성곱 하여 특성 맵(feature map) 구성



    - 필터와 이미지의 각 위치에 있는 값들을 곱하고 모든 행렬의 값을 더하여 구성
  - 스트라이드(stride) : 옆으로 이동하며 동일한 연산을 계속 진행
    - 스트라이드의 크기에 따라 출력값의 크기 변경
      - 이미지의 크기가 6×6, 필터가 3×3으로 구성
      - 스트라이드 값이 1이면 출력 이미지 크기는 4×4로 구성
  - 활성화함수로는 ReLU를 주로 사용

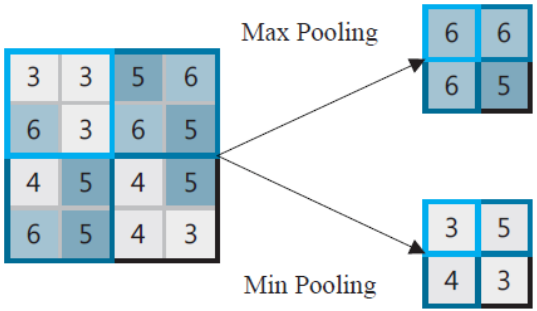
합성곱 신경망(CNN) 구현

- 합성곱 계층
  - 제로패딩(Zero Padding)
    - 필터의 크기와 스트라이드 값에 따라 출력 이미지 크기가 줄어드는 것 방지
    - 입력 이미지의 행렬의 상, 하, 좌, 우에 0을 채움

- 입력 이미지 6×6 크기에 제로 패딩을 사용하여 8×8 로 구성
- 3×3 크기의 필터를 한 칸씩 스트라이드
- 출력 이미지의 크기는 입력 이미지의 크기와 동일하게 구성됨

합성곱 신경망(CNN) 구현

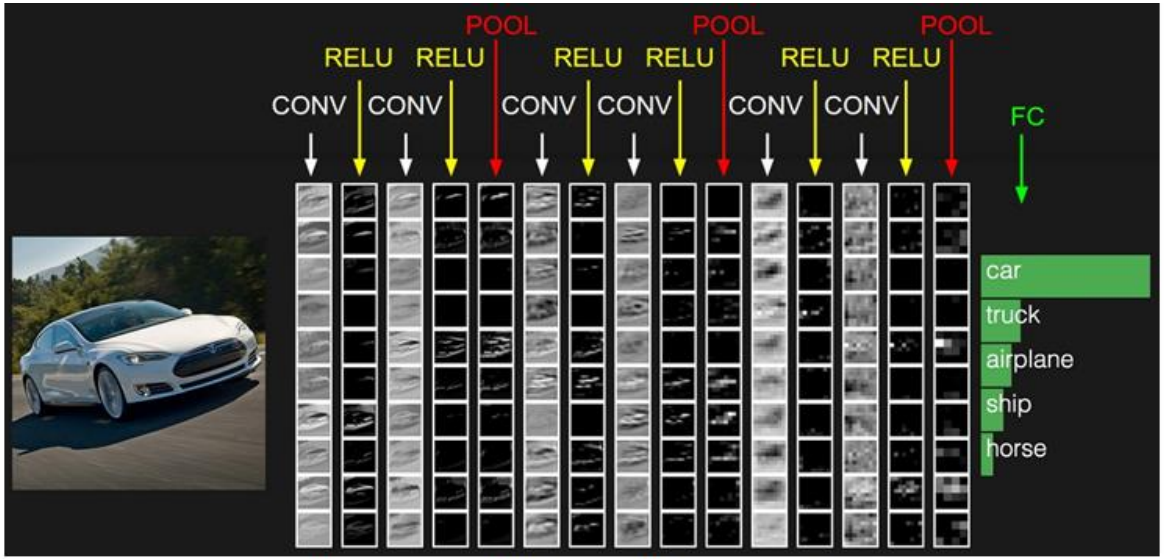
- 풀링 계층
  - 선택된 영역에서의 최솟값(Min Pooling), 최댓값(Max Pooling), 평균값(Average Pooling)을 풀링하여 이미지를 축소 처리
    - 차원을 축소함에 따라 연산량 감소
    - 과적합(Overfitting)을 방지
    - 영역 내에서의 특징을 가진 부분을 추출
  - 4×4 크기의 입력 이미지를 2×2 크기의 필터와 스트라이드 값을 2로 설정



  - 합성곱 신경망에서는 주로 최댓값 풀링을 사용

합성곱 신경망(CNN) 구현

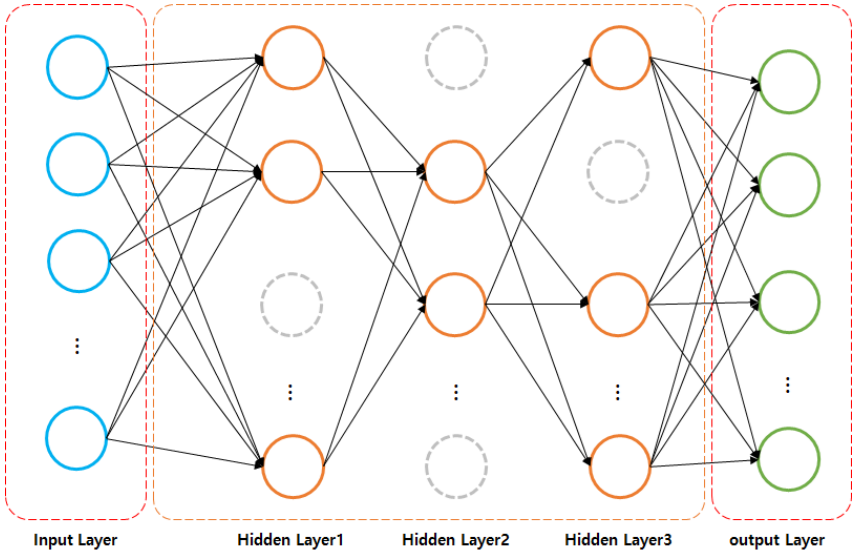
- 일반적인 신경망의 구조



http://cs231n.github.io/convolutional-networks/

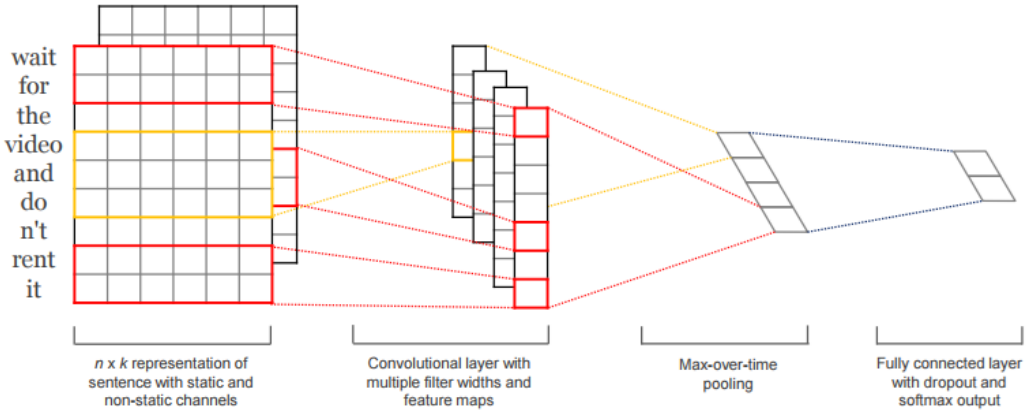
합성곱 신경망(CNN) 구현

- 드롭 아웃(DropOut)
  - 완전 결합 계층에서의 과적합(Overfitting)을 방지
  - 신경망에서의 뉴런들을 임의적으로 선택하여 버린 후 나머지 뉴런들에 대해서만 학습
  - 학습 시에는 드롭아웃 을 사용하고, 학습 이후 검증 시에는 모든 뉴런들을 사용하도록 드롭아웃을 사용하지 않는 것이 일반적인 방식



합성곱 신경망(CNN) 구현

- 자연어 처리에서의 합성곱 신경망 구조
  - 2014년 김윤 박사님의 논문 - Convolutional Neural Network for Sentence Classification
    - 단어들을 벡터화, 여러 필터 크기를 사용하여 합성곱 및 특성 맵을 구성
    - 최댓값 풀링, 드롭아웃과 Softmax를 통해 결괏값을 분류



"Convolutional Neural Network for Sentence Classification", Yoon Kim, 2014

합성곱 신경망(CNN) 구현

- 전처리 단계 - Step 1) 파라미터 설정, Word2Vec 로드
  - 파라미터 설정
    - 단어의 벡터 차원 수 -> embedding_dim : 100
    - 레이블 차원 수 -> class_sizes : 2
    - 문장 내 최대 단어 개수 -> max_sentence_length : 50
    - 합성곱 필터 사이즈 -> filter_sizes : [2 3 4]
    - 합성곱 특성 맵 개수 -> num_filters : 50
    - 학습 시 드롭아웃 변수 -> dropout_keep_prob : 0.5
    - 학습 횟수 -> num_epochs : 20
    - 패치 사이즈 -> batch_size : 1000
    - 검증을 위한 조건 -> evaluate_every : 150
    - 최적화 알고리즘 학습률 -> learn_rate : 0.001
  - Word2Vec 로드
    - Word2Vec.load() 함수를 통해 저장된 모델 로드
    - 모델에서의 vector_size 셋팅

```
72  def run_cnn(params) :
73      class_sizes = 2
74      max_sentence_length = int(params[1])
75      filter_sizes = np.array(params[2].split(','), dtype=int)
76      num_filters = int(params[3])
77      dropout_keep_prob = float(params[4])
78      num_epochs = int(params[5])
79      batch_size = int(params[6])
80      evaluate_every = int(params[7])
81      learn_rate = float(params[8])
82
83      model = Word2Vec.load(source_dir + w2v_file_name)
84      # word2vec 파일에서의 벡터 차원 수 계산
85      embedding_dim = model.vector_size
```

합성곱 신경망(CNN) 구현

- 전처리 단계 - Step 2) 데이터 준비
  - 트레이닝, 테스트 데이터 로드
  - 데이터 구조 셋팅
    - data size * word_length * embedding
  - 최대 단어 크기 셋팅 및 Word2Vec 변환
  - 라벨 셋팅 - One Hot Encoding

```
32  def data_setting(w2v_model, embedding_dim, class_sizes, max_word_length):
33      # 데이터 불러와서 문장의 총 개수 셋팅
34      train_data = load_data(source_dir + file_list[0])
35      train_size = len(train_data)
36      #print('train_size : ' + str(train_size))
37
38      test_data = load_data(source_dir + file_list[1])
39      test_size = len(test_data)
40      #print('dev_size : ' + str(dev_size))
41
42      # 데이터 구조 : 전체 문장 x 문장 내 단어 제한 수 x 벡터의 차원
43      train_arrays = np.zeros((train_size, max_word_length, embedding_dim))
44      test_arrays = np.zeros((test_size, max_word_length, embedding_dim))
45      # 정답의 구조 : 전체 문장 x 구분 수(긍정/부정)
46      train_labels = np.zeros((train_size, class_sizes))
47      test_labels = np.zeros((test_size, class_sizes))
48
49      for train in range(len(train_data)) :
50          # 각 문장의 단어를 벡터화 하고 문장 구성
51          train_arrays[train] = pre.max_word_length_word2vec(w2v_model, embedding_dim, max_word_length, train_data[train][0])
52          # 각 문장이 정답을 one-hot encoding으로 변경
53          train_labels[train] = label_value(int(train_data[train][1]), class_sizes)
54
55      for dev in range(len(test_data)) :
56          test_arrays[dev] = pre.max_word_length_word2vec(w2v_model, embedding_dim, max_word_length, test_data[dev][0])
57          test_labels[dev] = label_value(int(test_data[dev][1]), class_sizes)
58
59      return train_arrays, train_labels, test_arrays, test_labels
```

합성곱 신경망(CNN) 구현

- 전처리 단계 - Step 2) 데이터 준비
  - 데이터 로드
    - 문장을 읽어 탭으로 구분

```
# 파일을 읽어 각 문장을 탭으로 구분
def load_data(txtFilePath):
    #with open(txtFilePath,'r', encoding='UTF-8') as data_file:
    with open(txtFilePath,'r') as data_file:
        return [line.split('\t') for line in data_file.read().splitlines()]
```

  - 최대 단어 크기 셋팅 및 Word2Vec 변환
    - 최대 크기 단어 설정
      - 너무 크게 잡으면 성능 저하
      - 최대 단어 크기를 넘어가는 문장 내 단어 삭제
      - 최대 단어 크기보다 부족한 문장은 제로 패딩
    - 단어들에 대해 벡터 변환
    - Word2Vec 모델에 존재하지 않는 벡터 값은 제외

```
def max_word_length_word2vec(w2v_model, embedding_dim , max_word_length, word_list):
    # 문장 내 단어 제한 x 벡터 차원 수
    data_arrays = np.zeros((max_word_length, embedding_dim))

    # string 문장으로 들어오는 경우 split 처리
    if type(word_list) is str :
        word_list = word_list.split()

    # 단어를 벡터로 변경
    if len(word_list) > 0 :
        word_length = max_word_length if max_word_length < len(word_list) else len(word_list)

        for i in range(word_length):
            try :
                data_arrays[i] = w2v_model[word_list[i]]
            except KeyError :
                pass
    return data_arrays
```

합성곱 신경망(CNN) 구현

- 전처리 단계 - Step 2) 데이터 준비
  - 라벨 값 셋팅
    - One-Hot Encoding으로 설정
    - 긍정은 [0.1]
    - 부정은 [1.0] 으로 설정

```python
# 긍/부정에 대한 one-hot encoding
def label_value(code, size):
    code_arrays = np.zeros((size))
    # 부정인 경우 [1, 0]
    if code == 0:
        code_arrays[0] = 1
    # 긍정인 경우 [0, 1]
    elif code == 1:
        code_arrays[1] = 1

    return code_arrays
```

- 합성곱 신경망 모델 - Step 1) 모델 생성을 위한 변수 초기화
  - input_x : Word2Vec를 통해 문장에서의 각 단어별 벡터 데이터
    - 합성곱 계층에 들어가는 데이터는 4D 텐서로 표현되기 때문에 expand_dims을 통해 차원을 확장
    - [batch_size, sequence_length, embedding_size, 1] 형태
  - input_y : 긍정/부정 값에 대한 one-hot encoding된 데이터
  - dropout_keep_prob : 드롭아웃하지 않고 유지 할 노드의 비율

```python
class cnn_model(object):
    def __init__(self, sequence_length, num_classes, embedding_size, filter_sizes, num_filters):
        # 학습 데이터가 들어갈 플레이스 홀더 선언
        self.input_x = tf.placeholder(tf.float32, [None, sequence_length, embedding_size], name="input_x")
        self.input_y = tf.placeholder(tf.float32, [None, num_classes], name="input_y")
        self.dropout_keep_prob = tf.placeholder(tf.float32, name="dropout_keep_prob")
        self.expanded_input_x = tf.expand_dims(self.input_x, -1)
```

합성곱 신경망(CNN) 구현

- 합성곱 신경망 모델 - Step 2) 합성곱/풀링 레이어
  - 합성곱 레이어
    - [filter_size, embedding_size, 1, num_filters] 형태의 필터 와 expanded_input_x 입력값의 합성곱
    - 가로, 세로 1칸씩 이동하는 스트라이드 설정
    - 특성 맵(feature map) : sentence_length - filter_size + 1
    - 특성 맵의 개수인 num_filters 변숫값을 통해 [batch_size, sentence_length - filter_size + 1, 1, num_filters] 형태의 합성곱 텐서가 생성

```
# 각 필터별 합성곱 레이어 + 풀링 레이어 생성
pooled_outputs = list()
for i, filter_size in enumerate(filter_sizes):
    with tf.name_scope("conv-maxpool-%s" % filter_size):
        # 합성곱 레이어
        filter_shape = [filter_size, embedding_size, 1, num_filters]
        W = tf.Variable(tf.truncated_normal(filter_shape, stddev=0.1), name="W")
        b = tf.Variable(tf.constant(0.1, shape=[num_filters]), name="b")
        conv = tf.nn.conv2d(self.expanded_input_x, W, strides=[1, 1, 1, 1], padding="VALID", name="conv")
        h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")

        # 맥스 풀링 레이어
        pooled = tf.nn.max_pool(h, ksize=[1, sequence_length - filter_size + 1, 1, 1], strides=[1, 1, 1, 1], padding='VALID', name="pool")
        pooled_outputs.append(pooled)
```

합성곱 신경망(CNN) 구현

- 합성곱 신경망 모델 - Step 2) 합성곱/풀링 레이어
  - 풀링 레이어
    - 합성곱 레이어와 동일한 [1, sequence_length - filter_size + 1, 1, 1]인 커널
    - 가로, 세로 1칸씩 이동하는 스트라이드 설정
    - 최댓값 풀링을 적용하여 [batch_size, 1, 1, num_filters] 형태의 결과가 출력
    - Padding 옵션
      - “VALID” : 제로 패딩을 하지 않고 스트라이드에 따라 오른쪽의 행, 열 값이 무시
      - “SAME” : 제로 패딩을 사용하여 똑같은 크기의 차원이 리턴
    - pooled_outputs : filter_sizes에 따른 합성곱 계층 과 풀링 계층의 동일한 연산을 3번 수행
    - 3개의 풀링 데이터를 합쳐 [batch_size, num_filters_total] 형태의 Fully-Connected Layer 생성

```
# 풀링된 데이터 통합 및 차원 변경
num_filters_total = num_filters * len(filter_sizes)
self.h_pool = tf.concat(pooled_outputs, 3)
self.h_pool_flat = tf.reshape(self.h_pool, [-1, num_filters_total])
```

합성곱 신경망(CNN) 구현

- 합성곱 신경망 모델 - Step 2) 합성곱/풀링 레이어
  - 드롭 아웃 및 최종 출력 레이어
    - 학습 중에는 0.5로 절반을 랜덤으로 비활성성화 검증에서는 1.0으로 비활성화하지 않도록 설정
    - Softmax를 통해 최종 분류를 수행
    - 가중치의 초기화 : tf.contrib.layers.xavier_initializer 함수
      - 2010년 Glorot과 Bengio가 발표
      - Xavier는 입력값과 출력값의 난수를 선택하여 입력값의 제곱근으로 나누는 것
        - W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in)

```
31          # 드랍아웃 적용
32          with tf.name_scope("dropout"):
33              self.h_drop = tf.nn.dropout(self.h_pool_flat, self.dropout_keep_prob)
34
35          # Output Layer
36          with tf.name_scope("output"):
37              W = tf.get_variable("W", shape=[num_filters_total, num_classes], initializer=tf.contrib.layers.xavier_initializer())
38              b = tf.Variable(tf.constant(0.1, shape=[num_classes]), name="b")
39              self.scores = tf.nn.xw_plus_b(self.h_drop, W, b, name="scores")
40              self.predictions = tf.argmax(self.scores, 1, name="predictions")
41              self.result =  tf.nn.softmax(logits=self.scores, name="result")
```

합성곱 신경망(CNN) 구현

- 합성곱 신경망 모델 - Step 2) 합성곱/풀링 레이어
  - 드롭 아웃 및 최종 출력 레이어
    - 학습 중에는 0.5로 절반을 랜덤으로 비활성성화 검증에서는 1.0으로 비활성화하지 않도록 설정
    - Softmax를 통해 최종 분류를 수행
    - 가중치의 초기화 : tf.contrib.layers.xavier_initializer 함수
      - 2010년 Glorot과 Bengio가 발표
      - Xavier는 입력값과 출력값의 난수를 선택하여 입력값의 제곱근으로 나누는 것
        - W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in)

```
31          # 드랍아웃 적용
32          with tf.name_scope("dropout"):
33              self.h_drop = tf.nn.dropout(self.h_pool_flat, self.dropout_keep_prob)
34
35          # Output Layer
36          with tf.name_scope("output"):
37              W = tf.get_variable("W", shape=[num_filters_total, num_classes], initializer=tf.contrib.layers.xavier_initializer())
38              b = tf.Variable(tf.constant(0.1, shape=[num_classes]), name="b")
39              self.scores = tf.nn.xw_plus_b(self.h_drop, W, b, name="scores")
40              self.predictions = tf.argmax(self.scores, 1, name="predictions")
41              self.result =  tf.nn.softmax(logits=self.scores, name="result")
```

합성곱 신경망(CNN) 구현

- 합성곱 신경망 모델 - Step 3) 비용 / 정확도 계산
  - 비용 함수
    - 비용 함수 : softmax_cross_entropy_with_logits_v2
      - 최신 Tensorflow 버전에서는 tf.nn.softmax_cross_entropy_with_logits가 deprecated될 예정

```
43          # 비용 함수(오차, 손실함수) 선언
44          with tf.name_scope("loss"):
45              # v2 아닌 method는 deprecated 예정
46              self.cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=self.scores, labels=self.input_y))
47
48          # 정확도 계산
49          with tf.name_scope("accuracy"):
50              correct_predictions = tf.equal(self.predictions, tf.argmax(self.input_y, axis=1))
51              self.accuracy = tf.reduce_mean(tf.cast(correct_predictions, tf.float32), name="accuracy")
```

합성곱 신경망(CNN) 구현

- 합성곱 신경망 모델 학습 실행 - Step 1) 비용 / 정확도 계산
  - 비용 함수
    - 비용 함수 : softmax_cross_entropy_with_logits_v2
      - 최신 Tensorflow 버전에서는 tf.nn.softmax_cross_entropy_with_logits가 deprecated될 예정

```
43          # 비용 함수(오차, 손실함수) 선언
44          with tf.name_scope("loss"):
45              # v2 아닌 method는 deprecated 예정
46              self.cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=self.scores, labels=self.input_y))
47
48          # 정확도 계산
49          with tf.name_scope("accuracy"):
50              correct_predictions = tf.equal(self.predictions, tf.argmax(self.input_y, axis=1))
51              self.accuracy = tf.reduce_mean(tf.cast(correct_predictions, tf.float32), name="accuracy")
```

합성곱 신경망(CNN) 구현

- 합성곱 신경망 모델 학습 실행
  - tf.Graph().as_default()를 통해 그래프를 생성
    - tf.ConfigProto() : gpu_options.allow_growth 와 같은 GPU를 사용하는 만큼만 증가시키는 옵션들을 설정
    - 옵션들은 tf.Session(config=sess_config)에 파라미터로 넘겨 사용

```
115     with tf.Graph().as_default():
116         #sess_config = tf.ConfigProto(device_count = {'GPU': 0})
117         sess_config = tf.ConfigProto()
118         sess_config.gpu_options.allow_growth = True
119         sess = tf.Session(config=sess_config)
```

  - sess.as_default()를 사용하여 세션 범위 지정
    - 합성곱 신경망에 사용할 변수 초기화

```
121         with sess.as_default():
122             cnn = cnn_model(
123                 sequence_length=x_train.shape[1],
124                 num_classes=y_train.shape[1],
125                 embedding_size=embedding_dim,
126                 filter_sizes=filter_sizes,
127                 num_filters=num_filters)
```

합성곱 신경망(CNN) 구현

- 합성곱 신경망 모델 학습 실행
  - 최적화 함수 : AdamOptimizer
    - 비용 함수의 값이 최소화 되도록 minimize() 함수 사용
      - compute_gradients + apply_gradients 함수 사용과 동일
  - 모델과 파라미터를 저장하기 위해 tf.train.Saver() 사용
  - 변수들을 초기화하기 위해 tf.global_variables_initializer()를 수행

```
130             # 비용함수의 값이 최소가 되도록 하는 최적화 함수 선언
131             optimizer = tf.train.AdamOptimizer(learn_rate)
132             train_op = optimizer.minimize(cnn.cost, global_step=global_step)
133             #grads_and_vars = optimizer.compute_gradients(cnn.cost)
134             #train_op = optimizer.apply_gradients(grads_and_vars, global_step=global_step)
135
136             saver = tf.train.Saver()
137             sess.run(tf.global_variables_initializer())
```

합성곱 신경망(CNN) 구현

- 합성곱 신경망 모델 학습 실행
  - 학습을 위해 배치 트레이닝 방식을 사용
    - make_batch 함수를 구현하여 배치 사이즈만큼 학습할 수 있도록 리스트를 생성

```
61  def make_batch(list_data, batch_size):
62      num_batches = int(len(list_data)/batch_size)
63      batches = list()
64
65      for i in range(num_batches):
66          start = int(i * batch_size)
67          end = int(start + batch_size)
68          batches.append(list_data[start:end])
69
70      return batches
```

  - 배치별 트레이닝, 검증 작업 수행
    - 1,000건 단위로 배치를 생성
    - 학습 step 값이 150으로 나눠떨어질 때마다 테스트 데이터를 통해 학습된 모델에 대한 검증

```
168             # 배치별 트레이닝, 검증
169             for epoch in range(num_epochs):
170                 for len_batch in range(len(train_x_batches)):
171                     train_step(train_x_batches[len_batch], train_y_batches[len_batch])
172                     current_step = tf.train.global_step(sess, global_step)
173                     if current_step % evaluate_every == 0:
174                         dev_step(x_dev, y_dev, epoch)
```

합성곱 신경망(CNN) 구현

- 합성곱 신경망 모델 학습 실행
  - train_step 과 dev_step에서의 feed_dict 값의 차이는 dropout 값
  - 각 step 값과 정확도, 비용 값을 시각화를 위해 리스트로 저장
  - 학습 결과
    - 총 20회의 학습
    - 비용 값은 0.385에서 0.314로 감소
    - 정확도는 83.1%에서 86.7%로 증가

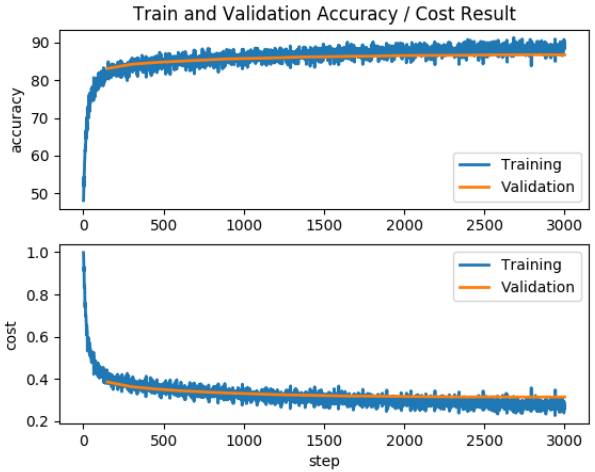
```
def train_step(x_batch, y_batch):
    feed_dict = {
      cnn.input_x: x_batch,
      cnn.input_y: y_batch,
      cnn.dropout_keep_prob: dropout_keep_prob
    }
    _, step, cost, accuracy = sess.run([train_op, global_step, cnn.cost, cnn.accuracy], feed_dict)
    train_x_plot.append(step)
    train_y_accracy.append(accuracy * 100)
    train_y_cost.append(cost)
    #print("Train step {}, cost {:g}, accuracy {:g}".format(step, cost, accuracy))


def dev_step(x_batch, y_batch, epoch):
    feed_dict = {
      cnn.input_x: x_batch,
      cnn.input_y: y_batch,
      cnn.dropout_keep_prob: 1.0
    }
    step, cost, accuracy, dev_pred = sess.run([global_step, cnn.cost, cnn.accuracy, cnn.predictions], feed_dict)
    valid_x_plot.append(step)
    valid_y_accuracy.append(accuracy * 100)
    valid_y_cost.append(cost)
    print("Valid step, epoch {}, step {}, cost {:g}, accuracy {:g}".format((epoch+1), step, cost, accuracy))
```

합성곱 신경망(CNN) 구현

- 합성곱 신경망 모델 학습 실행
  - 학습과 검증에서의 정확도와 비용에 대한 시각화
    - 이미지 파일로 저장

```
178     # 학습 / 검증에서의 정확도와 비용 시각화
179     plt.subplot(2,1,1)
180     plt.plot(train_x_plot, train_y_accracy, linewidth = 2, label = 'Training')
181     plt.plot(valid_x_plot, valid_y_accuracy, linewidth = 2, label = 'Validation')
182     plt.title("Train and Validation Accuracy / Cost Result")
183     plt.ylabel('accuracy')
184     plt.legend()
185
186     plt.subplot(2,1,2)
187     plt.plot(train_x_plot, train_y_cost, linewidth = 2, label = 'Training')
188     plt.plot(valid_x_plot, valid_y_cost, linewidth = 2, label = 'Validation')
189     plt.xlabel('step')
190     plt.ylabel('cost')
191     plt.legend()
192
193     # 이미지로 저장
194     plt.savefig(source_dir + fig_file_name)
```



합성곱 신경망(CNN) 구현

- 합성곱 신경망 모델 학습 실행
  - 학습이 완료된 모델 저장

```
175             # 모델 저장
176             saver.save(sess, "./cnn_model/model.ckpt")
```

checkpoint : 이름으로 저장된 체크 포인트 파일 기록
  > model_checkpoint_path: "model.ckpt"
  > alll_model_checkpoint_paths: "model.ckpt"
model.ckpt.data-00000-of-00001 / model.ckpt.index : 학습된 파라미터 저장
model.ckpt.meta : 모델의 그래프 구조 저장(variable, operations, collections 등)

합성곱 신경망(CNN) 구현

- 합성곱 신경망 모델 전반적 구조