Word Vectors, Advanced RNN, and Embedding Visualization

Introduction to Word Embeddings

텍스트를 처리하는 전통적인 NLP 접근법 중 하나는 문장의 각 단어를 정수 ID로 표시하는 방식이다. 앞 장에서 했던 것처럼 예를 들어 ‘agriculture’라는 단어는 정수 3452로, ‘farm’은 12, ‘AI’는 150, ‘deep-learning’은 0으로 매핑할 수 있다.

이런 표현을 사용함으로써 기초적인 NLP 작업에서는 실제로 훌륭한 결과를 얻었지만 몇 가지 중요한 문제가 내제되어 있다. 먼저 이러한 고립된 표현을 사용함으로써 단어 내에 숨어 있는 모든 의미를 잃어버리게 되고 따라서 단어 사이의 의미적 근접성과 관련 정보를 놓치게 된다. 예를 들어 한 문장 안에서 agriculture와 farm, AI와 deep-learning은 서로 밀접한 단어인 반면, deep-learning과 farm은 서로 별 관련이 없다. 이러한 내용은 임의의 정수 ID에 반영되지 않는다.

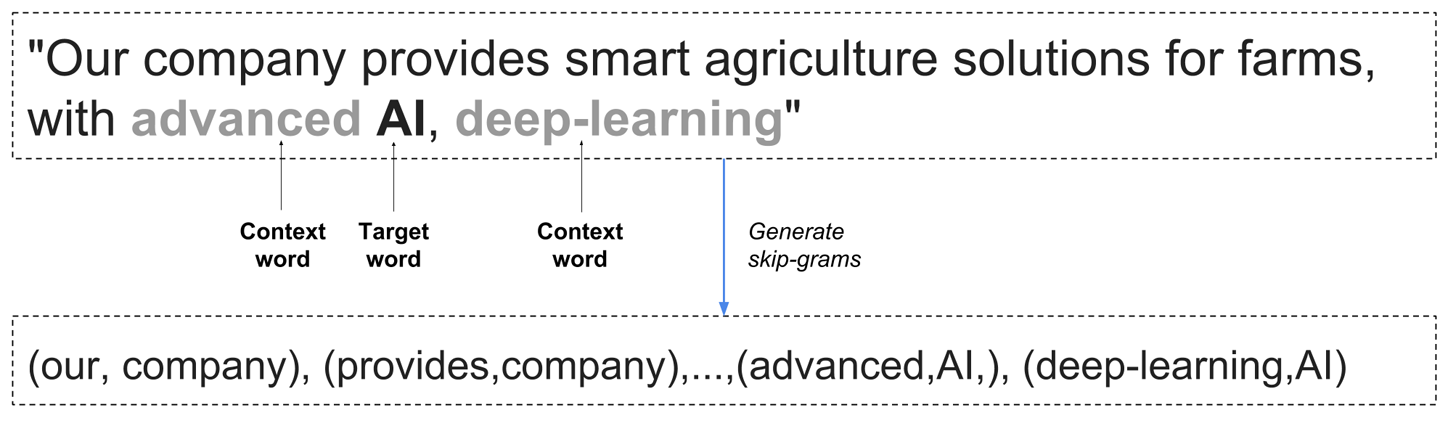
MNIST 데이터처럼 이미지라면 경우가 다르다. 물론 이미지의 차원도 커질 수 있지만, 픽셀 값이라는 표현 자체에 이미 어떠한 의미가 부호화되어 있으므로, 매우 고밀도의 표현이라 할 수 있다. 따라서 의미를 지니고 있는 단어에 대해서도 고밀도 벡터 표현을 사용하고자 한다. 하지만 어떻게 고밀도 단어 벡터를 얻을 수 있을까?

Chpater 5에서는 레이블이 붙어 있는 데이터를 사용해서 특정 작업을 수행하는 단어 벡터에 지도 학습을 적용했다. 하지만 개인 또는 단체가 항상 이렇게 할 수 있는 것은 아니다. 일일이 텍스트에 레이블을 붙이거나 쓸 만한 레이블이 붙어 있는 데이터를 확보하는 것은 자원, 시간, 노력의 측면에서 꽤나 비용이 큰 일이다. 반면 레이블이 붙어 있지 않은 대량의 데이터를 얻는 일은 그렇게 어려운 일이 아니다. 따라서 레이블이 없는 대량의 데이터를 사용해 비지도 학습으로 단어 표현을 학습하는 방법이 있다면 더 좋을 것이다.

사실 여전히 사용되는 전통적인 NLP건 신경망을 사용하는 새로운 방식이건, 단어 임베딩을 비지도 학습하는 방법은 여러 가지가 있다. 어떤 방법이든 그 핵심은 분포 가설(distridution hypothesis)에 의지하며, 이 가설은 언어학자 John Rupert Firth의 유명한 말로 쉽게 설명되곤 한다. “단어는 포함된 문맥 속에서 이해할 수 있다.” 즉 비슷한 맥락에서 함께 나타나는 경향이 있는 단어들은 비슷한 의미를 가지는 경향이 있다.

word2vec

word2vec은 잘 알려진 비지도 방식의 단어 임베딩 접근법이다. 실제로는 단어의 표현을 알기 위해 단어가 나타난 문맥을 활용하는 여러 알고리즘의 모음 같은 것이다. 이 구현은 단어의 입력이 주어지면 skip-gram이라는 것을 사용해 단어의 문맥을 예측하는 모델을 학습한다.



텍스트에서 skip-gram 생성하기

skip-gram wrod2vec 모델에서는 입력 단어를 기반으로 문맥을 예측하는 모델을 학습한다. 이건 바로 (our, company), (provides, company), (advanced, AI), (deep-learning, AI) 식으로 학습 데이터와 레이블의 쌍을 만든다는 의미이다.

데이터에서 추출한 이와 같은 쌍에 더해 ‘가짜’ 쌍도 추출한다. 주어진 입력 단어에 대해 문맥에 무작위 노이즈 단어들을 추가한다는 뜻인데 이를 negative sampling이라고 부른다. 이렇게 진짜 쌍과 가짜 쌍을 섞어 학습 데이터와 레이블을 만들고 이 둘을 구분할 수 있는 이진 분류기를 학습시킨다. 이 분류기에서 학습 가능한 매개변수는 벡터 표현, 즉 단어 임베딩이다. 이 벡터를 튜닝하여 문맥에 맞는 단어와 무작위로 추출된 단어의 차이를 설명할 수 있는 이진 분류기를 만들어낸다.

skip-gram

(word2vec.py)

데이터를 준비하고 skip-gram을 추출하는 데서 시작한다. 데이터는 매우 짧은 두 종류의 ‘문장’으로 구성되는데 하나는 홀수 숫자로, 다른 하나는 짝수 숫자로 이루어져 있다.

Embeddings int TensorFlow

단어 임베딩은 단어를 벡터값으로 매핑하는 룩업 테이블로도 볼 수 있는데, 손실 함수의 값을 최소화하도록 학습하는 과정에서 최적화된다. 여기서는 비지도 방식을 고려한 손실 함수를 사용한다

The Noise-Contrastive Estimation (NCE) Loss Function

skip-gram을 소개할 때, 진짜 문맥-타깃 단어 쌍 외에 무작위 문맥 단어를 사용하여 만든 가짜 잡음 쌍도 삽입한다고 말했다. 좋은 단어의 표현을 알 수 있도록, 이 둘을 구분하는 방법을 학습시켜야 한다. 무작위 잡음 쌍을 일일이 만들어도 되긴 하지만, 다행이 텐서플로는 이 작업에 사용할 수 있도록 설계된 잡음 대비 추정(noise-constrative estimation)이라는 쓸 만한 손실 함수를 제공한다. tf.nn.nce\_loss()를 사용하면 손실을 계산할 때 잡음 표본을 자동으로 만든다.

Learning Rate Decay

경사 하강법 최적화는 손실 함수를 최소화하는 방향으로 조금씩 이동하면서 가중치를 조정한다. learning\_rate 하이퍼파라미터는 이 이동하는 크기를 결정한다. 모델을 경사 하강법으로 학습하는 과정에서는 일반적으로 이 값의 크기를 점차 감소시켜, 매개변수 공간상의 좋은 위치로 접근할 때 최적화 프로세스가 ‘안정화’되도록 한다.

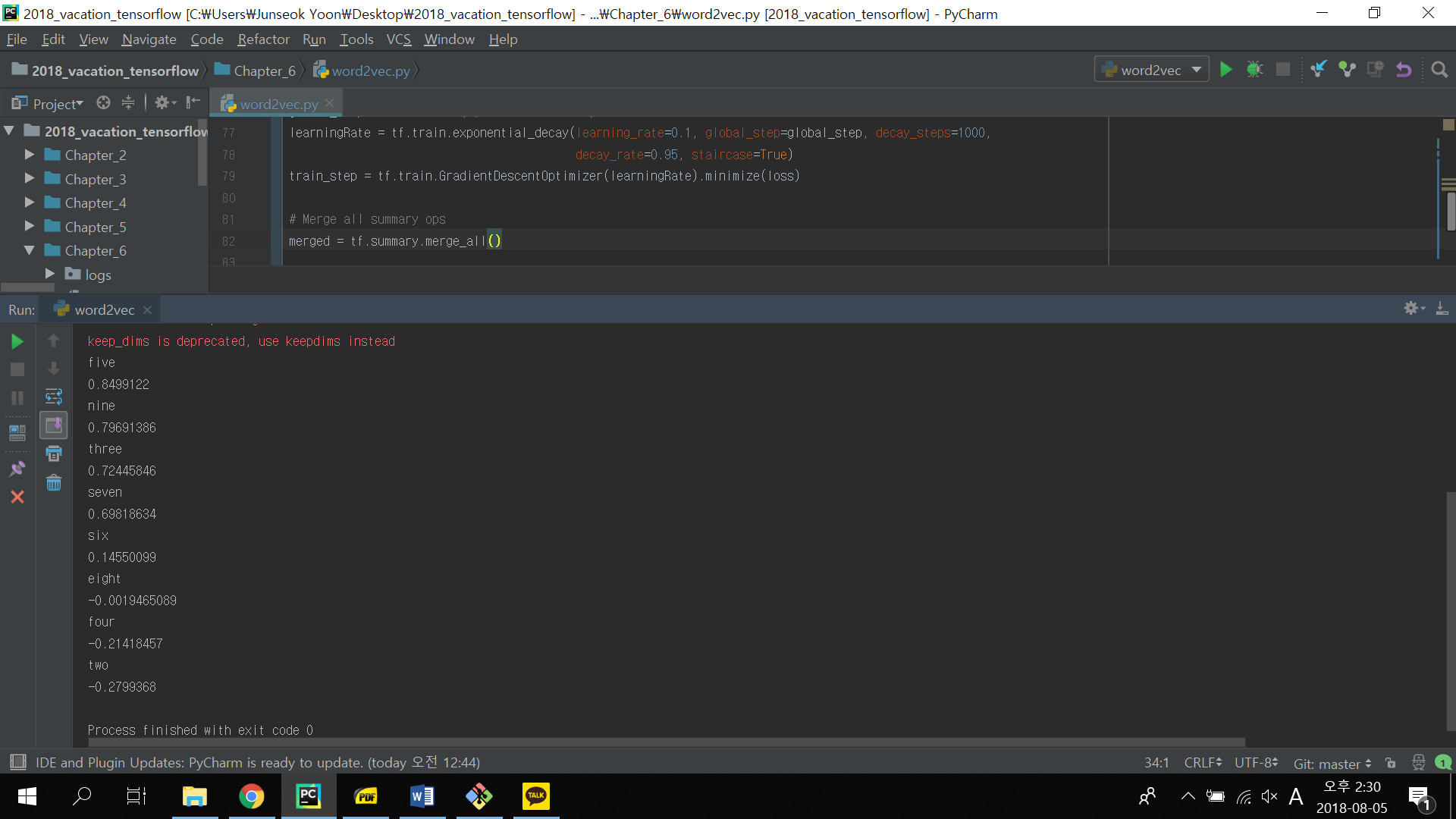
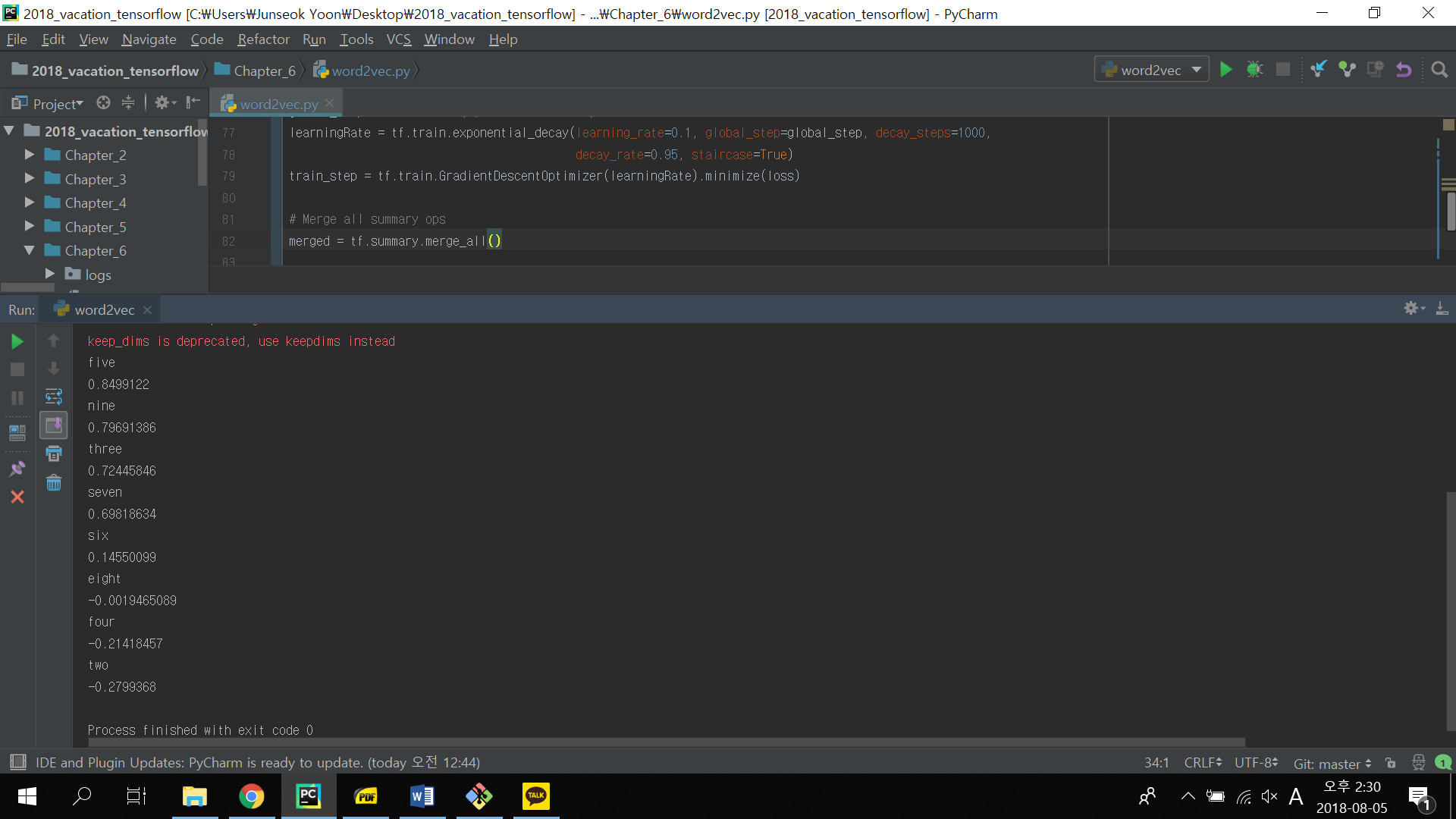
tf.train.exponential\_decay()는 학습률을 지수적으로 감소시키는데, 감소의 형태를 결정하는 몇 개의 하이퍼파라미터가 있다. 여기의 예제레서는 1,000단계마다 감소시키며 감소되는 학습률은 계단 함수의 형태를 따른다.

Training and Visualizing with TensorBoard

먼저 TSV 형식의 메타데이터 파일을 생성한다. 이 파일은 임베딩 벡터를 연관 레이블이나 이미지와 연결한다. 이 예제에서 각 임베딩 벡터는 그 벡터가 나타내는 단어의 레이블을 가지고 있다. 그런 다음 tensorboard에 임베딩 변수를 지정하고 이 변수를 메타데이터 파일에 연결한다. 마지막으로 최적화가 끝난 후 세션을 닫기 전에 단어 임베딩 벡터를 단위 길이로 정규화하는 표준 후처리 과정을 진행한다.

Checking Out Our Embeddings

결과로 얻은 단어 벡터를 간단히 살펴보자. 단어 ‘one’을 선택해 다른 모든 단어 벡터를 얼마나 가까우냐에 따라서 내림차순으로 정렬한다.

 벡터 내적의 결과로 보면, 홀수를 나타내는 단어 벡터가 ‘one’에 가깝고, 짝수를 나타내는 단어 벡터는 ‘one’에 가깝지 않다는 것을 알 수 있다. 임베딩된 벡터를 학습하여 짝수와 홀수를 구분할 수 있게 되었다. 홀수와 짝수 각각에 속한 벡터들은 서로 멀리 분리되어 각 단어가 나타내는 맥락을 구성한다.

Pretrained Word Embeddings and Advanced RNN

단어 임베딩은 텍스트 처리를 위한 딥러닝 모델의 강력한 구성 요소이다. 많은 응용프로그램에서 널리 사용되는 접근 방법은 대량의 텍스트 데이터상에서 word2vec와 같은 방법으로 단어 벡터를 학습한 다음. 이 벡터를 사용해 지도 학습 방식의 문서 분류와 같은 후속 작업을 수행하는 것이다.

앞에서는 단어 벡터를 비지도 방식으로 맨 처음부터 학습시켰다. 이렇게 하려면 일반적으로 위키백과의 내용이나 웹 페이지 같은 대량의 말뭉치들이 필요하다. 실전에서는 대량의 말뭉치들을 사용해 학습된 pretrained word embedding을 쓰는 경우가 종종 있는데 앞에서 본 학습시켜 사용한 단어 임베딩 방식과 많이 닮았다.

이 절에서는 간단한 텍스트 분류 작업을 통해 tensorflow에서 pretrained word embedding을 사용하는 방법을 보여준다. 또한 자연어이해를 위한 최신 딥러닝 응용프로그램에서 사용되는 더 유용하고 강력한 양방향 RNN 계층과 게이트 정류 유닛(GRU) 셀을 소개한다.

Pretrained Word Embeddings

웹 데이터를 기반으로 학습된 단어 벡터를 가져와서 텍스트 분류 작업에 통합하는 방법을 보여준다. 임베딩 방법은 GloVe라는 방법이다.

(GRU\_pretrained\_GloVe.py)

내려받은 사전 학습된 GloVe 임베딩 속의 어휘는 220만 개이며 간단한 예제는 9개에 불과하다. 9개 단어에 대해서만 GloVe 벡터를 가져온다.

GloVe 파일을 한 줄씩 뒤져가면서 필요한 단어 벡터를 가지고 와서 정규화한다. 필요한 9개 단어를 추출하고 나면 처리를 중단하고 루프에서 빠져나온다. 이 함수의 결과는 각 단어로부터 대응되는 벡터에 매핑되는 딕셔너리이다.

예제의 임베딩은 embedding\_plcaholder의 값으로 초기화되는데 assign() 함수를 사용해 embeddings 변수에 초깃값을 대입한다. trainable=True로 설정하여 tensorflow가 현재 작업에 맞춰 단어 벡터의 값을 최적화하여 갱신하도록 지정한다. 하지만 어떤 경우에는 trainable=False로 설정하여 갱신되지 않도록 설정하는 것이 나을 때도 있다. 예를 들어 충분한 양의 레이블된 데이터가 확보되지 않은 상황이거나 단어 벡터가 이미 충분히 ‘훌륭’해서 이후의 패턴을 찾아내는 데 문제가 없다는 확신이 있는 경우가 이에 해당된다.

단어 벡터를 학습에 완전히 통합하는 데는 한 단계가 더 필요하다. embedding\_placeholder에 embedding\_matrix의 값을 넣는 과정이다.

Bidirectional RNN and GRU Cells

양방향 RNN 계층은 RNN 계층을 단순히 확장한 것이다. 양방향 RNN 계층의 기본적인 형태는 왼쪽에서 오른쪽의 시퀀스를 읽는 계층과 오른쪽에서 왼쪽으로 시퀀스를 읽는 두 개의 평범한RNN 계층으로 구성된다. 각 RNN 계층은 왼쪽에서 오른쪽으로의 벡터 과 오른쪽에서 왼쪽으로의 벡터 의 숨겨진 표현을 산출한다. 두 벡터는 이후 하나의 벡터로 합쳐진다. 이러한 표현의 가장 큰 이점은 양방향으로부터의 단어의 문맥을 찾아낼 가능성인데 이를 통해서 자연어와 텍스트에 내재된 의미를 더 잘 이해할 수 있게 된다. 품사를 태깅할 때 문장에서 각 단어의 예측 태그를 만드는 예를 들어보자. 주어진 단어의 품사 태그를 예측하기 위해서는 그 단어의 앞과 뒤 양방향의 주위 단어의 정보를 알아야 한다.

게이트 정류 유닛(GRU) 셀은 LSTM 셀의 간소화된 유형 중 하나이다. 여기에도 기억 메커니즘이 있지만 LSTM에 비해서 꽤 적은 수의 매개변수를 가진다. 사용 가능한 데이터가 적을 때 흔히 사용되며 계산도 더 빠르다.

tensorflow는 양방향 계층을 위한 dynamic\_rnn()의 확장판인 tf.nn.bidirectional\_dynamic\_rnn()을 제공한다. 각각 왼쪽에서 오른쪽 및 오른쪽에서 왼쪽 벡터에 해당하는 cell\_fw와 cell\_bw RNN 셀이 필요하다. 여기서는 정방향 및 역방향 표현을 찾는 데 GRUCell()을 사용하고 DropoutWrapper()를 사용해 정규화를 위한 드롭아웃을 추가한다.

적당한 축을 기준으로 tf.concat()을 사용해 정방향 및 역방향 상태 벡터를 합치고 소프트맥스 함수를 적용한 선형 계층을 추가한다. 학습 준비가 끝나면, embedding\_placeholder에 embedding\_matrix를 밀어 넣어 초기화한다. 이 작업을 tf.global\_variable\_initializer()를 호출한 다음에 수행하는 것이 매우 중요하다. 순서가 바뀌면 초기화 함수가 사전 학습된 벡터를 초기화해버린다.