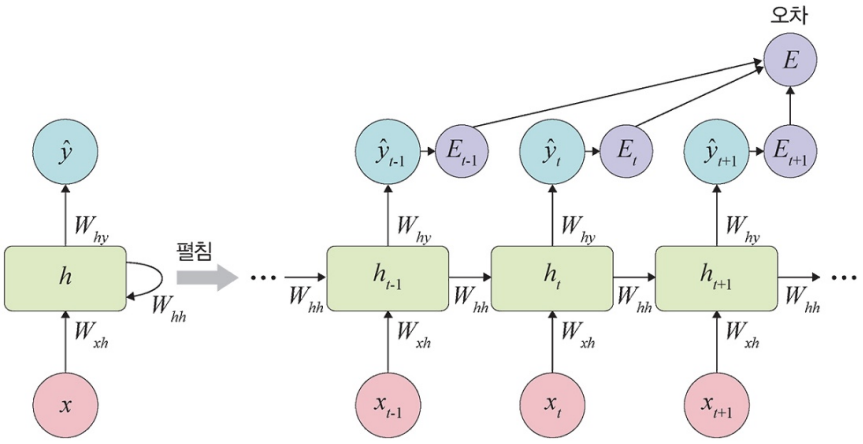
💛

4. RNN 구조

4-0. Intro

RNN

- 은닉층 노드들이 연결되어 이전 단계 정보를 은닉층 노드에 저장할 수 있도록 구성된 신경망
  - 과거 정보와 현재 정보를 반복해서 모두 반영

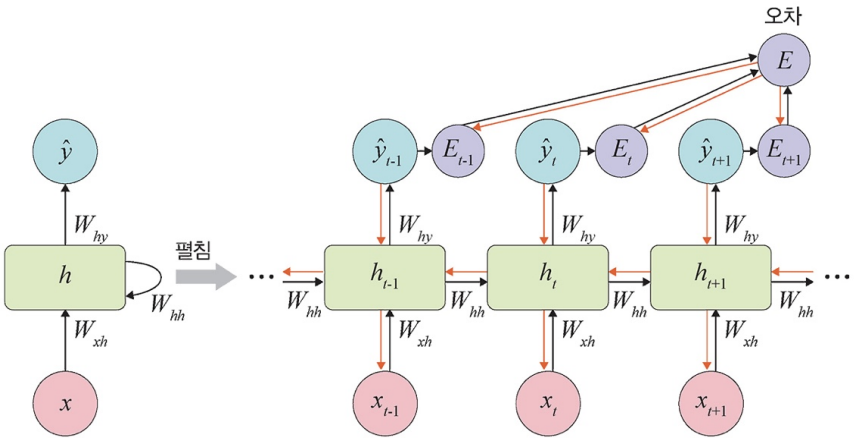


RNN 구조

- **RNN의 가중치**
  - 입력층에서 은닉층으로 전달되는 가중치(W_{xh})
  - t 시점의 은닉층에서 $t+1$ 시점의 은닉층으로 전달되는 가중치(W_{hh})
  - 은닉층에서 출력층으로 전달되는 가중치(W_{hy})

❗ 가중치들은 모든 시점에 서로 동일함

- **RNN의 계산**
  - 순방향 학습



순방향 학습

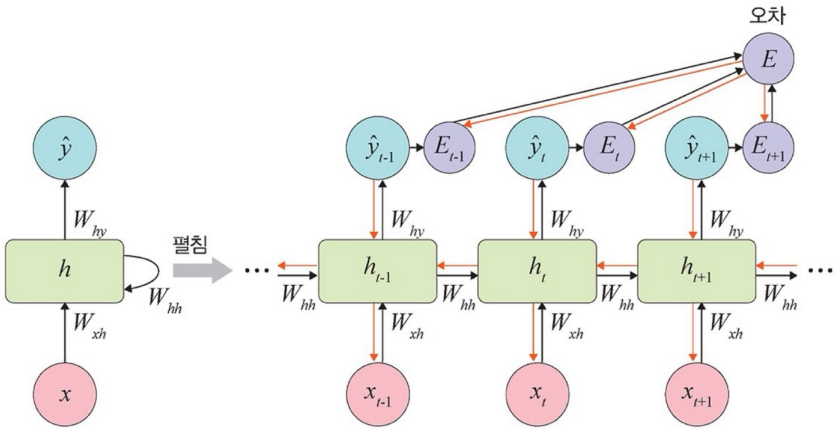
1. 은닉층
    - 이전 은닉층 * 은닉층 → 은닉층 가중치 + 입력층 → 은닉층 가중치 * (현재) 입력값
    - 활성화 함수로 주로 `tanh` 함수를 활용
    - 수식

      $$h_t = tanh(\hat{y}_t)$$

      $$\hat{y}_t = W_{hh} * h_{t-1} + W_{xh} * x_t$$
2. 출력층
    - 은닉층 → 출력층 가중치 * 현재 은닉층
    - 활성화 함수로 주로 `softmax` 함수를 활용
    - 수식

      $$\hat{y}_t = softmax(W_{hy} * h_t)$$
3. 오차(E)
    - 각 단계마다 실제 값(y_t)과 예측 값($\hat{y}_t$)으로 오차(⇒ 평균 제곱 오차)를 이용하여 측정

◦ 역방향 학습(역전파)



역방향 학습

- BPTT를 이용하여 모든 단계마다 처음부터 끝까지 역전파
  - BPTT(Back-Propagation Through Time): 각 단계(t)마다 오차 측정 후 이전 단계로 전달된 것
  - 이전에 계산된 오차를 통해 W_{xh}, W_{hh}, W_{hy} 및 bias를 업데이트
- BPTT는 오차가 멀리 전파될 때 계산량이 많아지고 전파되는 양이 점차 적어지는 문제가 발생

  → 기울기 소멸 문제

  → 이를 보완하기 위해 오차를 몇 단계까지만 전파시키는 생략된-BPTT를 활용하거나 근본적으로는 LSTM 및 GRU를 활용

4-1. RNN 셀 구현

데이터 전처리

torchtext

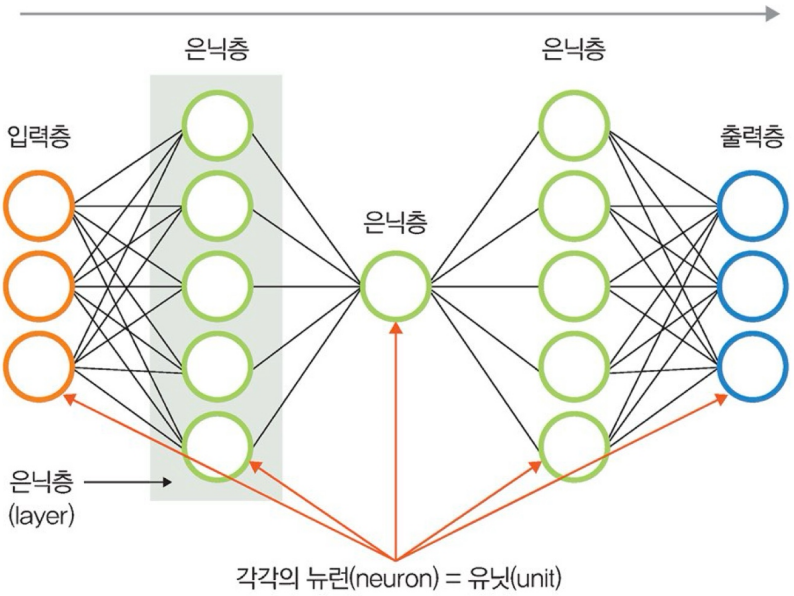
- 자연어 처리(NLP) 분야에서 사용하는 데이터로더
- 파일 가져오기, 토큰화, 단어 집합 생성, 인코딩, 단어 벡터 생성 등의 작업을 지원
- torchtext.legacy.data.`Field`
  - 데이터 전처리를 위해 사용됨
  - lower: 대문자 → 소문자
  - fix_length: 고정된 길이의 데이터를 얻을 수 있음
  - batch_first: 신경망에 입력되는 텐서의 첫 번째 차원 값이 배치 크기가 되도록 함
  - sequential: 데이터에 순서가 있는지를 나타내는 파라미터

단어 사전 생성

- Field.`build_vocab()`
  - data: 단어 사전을 생성할 데이터셋
  - max_size: 단어 집합의 크기 → 단어 집합에 포함되는 어휘 수
  - min_freq: 훈련 데이터셋에서 특정 단어의 최소 등장 횟수
  - vectors: 임베딩 벡터(= 워드 임베딩의 결과로 나온 벡터) 지정

data loading

- 데이터를 메모리에 올리는 작업
- 은닉층의 유닛 개수



은닉층과 뉴런/ 유닛

  - 일반적으로 계층(layer)의 유닛 개수를 늘리는 것보다 계층 자체 개수를 늘리는 것이 성능 향상에는 더 도움이 됨
    - 은닉층 개수의 증가는 인공 신경망이 비선형 문제를 좀 더 잘 학습할 수 있도록 함
    - 층 안에 포함된 뉴런 자체는 가중치와 바이어스를 계산하는 용도로 사용됨
  - 그러나 최적의 은닉층 개수와 유닛 개수를 찾는 것은 매우 어려움
    - 많은 경우 실제 필요한 개수보다 더 많은 층과 유닛을 구성한 후 과적합이 발생하지 않도록 이들의 개수를 조정해 나가는 방식을 택함

- `BucketIterator`
  - 데이터로더와 같은 역할을 수행

    → 배치 크기 단위로 값을 차례대로 꺼내 메모리로 가져오고 싶을 때 사용
  - 비슷한 길이의 데이터를 한 배치에 할당하여 패딩을 최소화시켜 줌
  - **parameter>**

```
torchtext.legacy.data.BucketIterator.splits(
(train_data, valid_data, test_data), batch_size=BATCH_SIZE, device=device)
                ⓐ                        ⓑ                  ⓒ
```

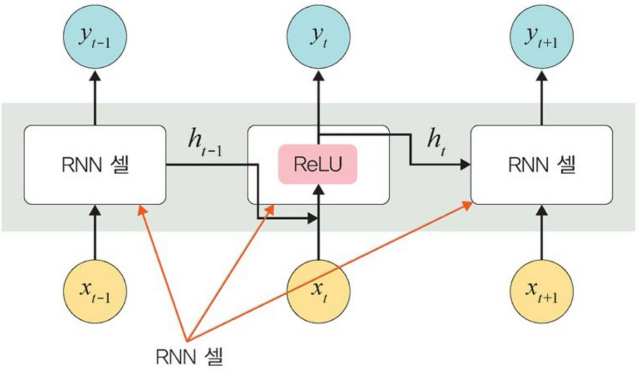
ⓐ 첫 번째 파라미터: 배치 크기 단위로 데이터를 가져올 데이터셋

ⓑ batch_size: 한 번에 가져올 데이터 크기(배치 크기)

ⓒ device: 어떤 장치(CPU 혹은 GPU)를 사용할지 지정

모델링

- 은닉층 상태



- 옵티마이져, 손실 함수
  - `torch.nn.CrossEntropyLoss()`
    - 다중 분류에 사용
    - `nn.LogSoftmax`와 `nn.NLLLoss` 연산의 조합

4-2. RNN 계층 구현

- `torch.nn.RNN()`
  - RNN 계층 구현

- embeded_dim: 훈련 데이터셋의 특성(feature) 개수
- hidden_dim: 은닉 계층의 뉴런(유닛) 개수
- num_layers: RNN 계층의 개수
- batch_first
  - 입력 데이터의 형태 → (시퀀스 길이, 배치 크기, 특성 개수)
  - True로 설정 시 배치 크기가 가장 앞으로 오게 됨