In [1]:

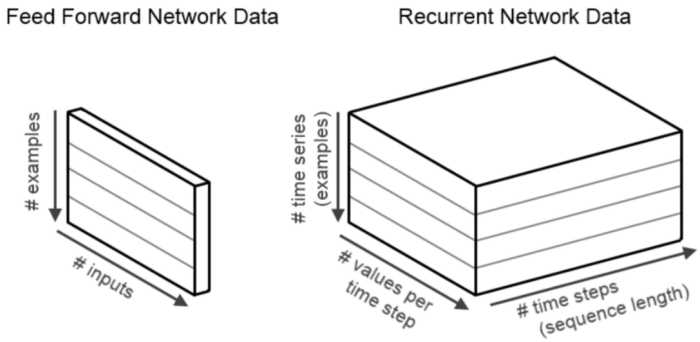
```
# 텐서플로우 docs
# >> https://tensorflowkorea.gitbooks.io/tensorflow-kr/content/g3doc/api_docs/python/array_ops.html
```

In [2]:

```
#  참고 자료 주소
# 참고 : http://colah.github.io/posts/2015-08-Understanding-LSTMs/
# 참고 : https://www.quora.com/In-LSTM-how-do-you-figure-out-what-size-the-weights-are-supposed-to-b
# 추천 : http://www.felixgers.de/papers/phd.pdf
# 성능을 증가: https://arxiv.org/pdf/1503.04069.pdf
# 망각 게이트의 bias를 1로 설정하는 팁: http://proceedings.mlr.press/v37/jozefowicz15.pdf
# 출처: https://dgkim5360.tistory.com/entry/understanding-long-short-term-memory-lstm-kr [개발새발로
```

```
ML = input data => 2- dimensional data  => [num_ex , inputDim]
Row :  데이터 개수
Col :  feature 개수 (차원? )

RNN = input data => 3- dimensional data  => [num_ex ,time Series_len,inputDim]
time Series_len : batch_size
# 샘플(sample)수          ---> 첫번째 축
# 시간 단계(time steps)수 ---> 두번째 축
# 특징(Features)수        ---> 세번째 축
```

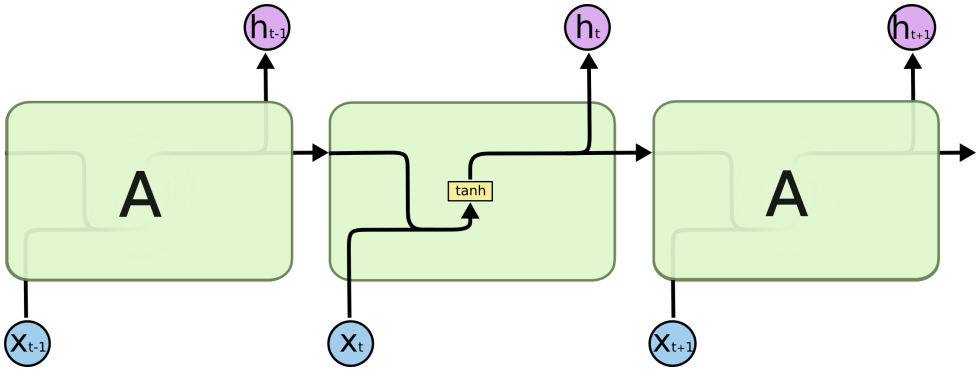


RNN_

RNN의 반복 모듈이 단 하나의 layer를 갖고 있는 표준적인 모습

기본적으로 RNN을 지식을 깔고 가야해

- time Series 개념을 가짐

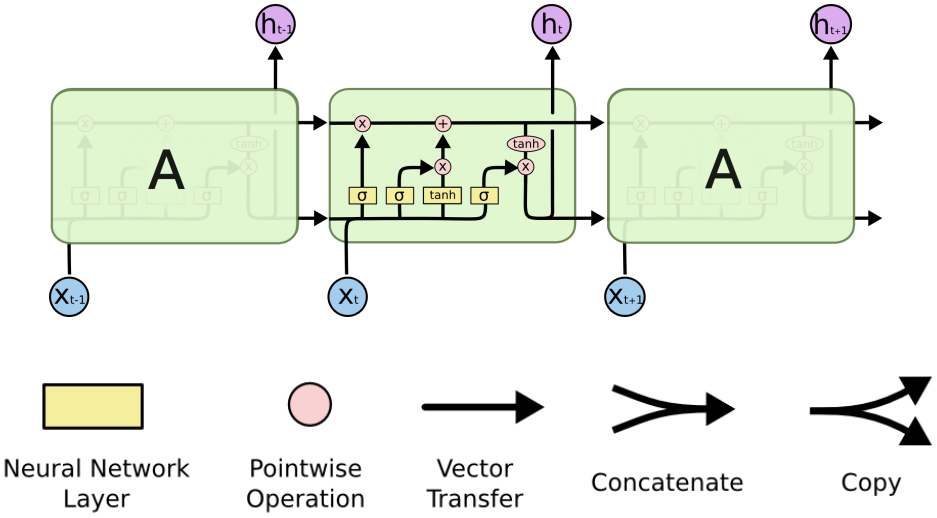


Long Short-Term Memory (LSTM) 이해하기

- RNN의 경우 긴 시간에 의존 => 이론상으로는 잘 다룬다 하지만 현실은 NO => 엉뚱한 예측(장기 의존성 문제)
- LSTM유닛은 여러 개의 게이트(gate)가 붙어있는 셀(cell)로 이루어져있음

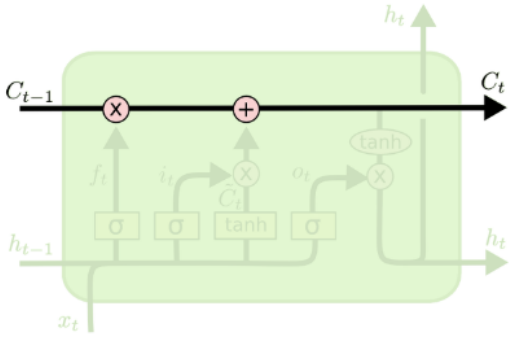
  \+ 이 셀의 정보를 새로 저장/셀의 정보를 불러오기/셀의 정보를 유지하는 기능

- LSTM => 핵심은 Cell state
- LSTM은 긴 의존 기간의 문제를 피하기 위해 명시적으로(explicitly) 설계
- RNN과 같은 체인과 구조 BUT 반복 모듈은 다르다.=> LSTM은 4개의 상호작용 layer 존재
- 의문) 기억력을 오래 보존하는 것이 목적인데 왜 LSTM은 망각 게이트 가지는가 => 때론 잊는 것이 좋을 때가 있기 때문 => 기억을 싹 지우고 새로 시작하는 것이 더 정확



Cell state ** (중요)

- Cell : 셀에 연결된 게이트의 값을 보고 무엇을 저장할지, 언제 정보를 내보낼지, 언제 쓰고 언제 지울지를 결정
- 작은 linear interaction만을 적용시키면서 전체 체인을 계속 구동
  => 뭔가를 더하거나 없앨 수 있는 능력(=gate)이 있는데, gate 구조에서 제어



LSTM의 cell state

Gate

- 3개의 Gate(입력/출력/망각 게이트)
- Gate는 정보가 전달방법 => **sigmoid layer x pointwise**
- 이 게이트가 열리거나(1) 닫히는(0) 디지털이 아니라 아날로그라는 점 주의
- 즉, 각 게이트는 0에서 1사이의 값을 가지며 게이트의 값에 비례해서 여러 가지 작동
- 게이트는 언제 신호를 불러올지/내보낼지/유지할지를 학습하며 이 학습과정은 출력의 오차를 이용한 경사하강법(gradient descent)을 사용 <이부분 조금더 봐야겠음 >

- 각 게이트가 갖는 값, 즉 게이트의 계수(또는 가중치, weight)는 은닉층의 값과 같은 원리로 학습

LSTM 구조

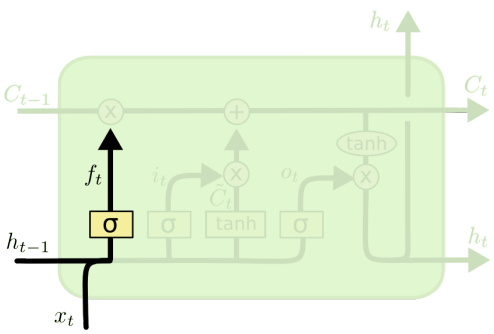
- 반복 모듈에는 4개의 레이어가 존재
- 3개의 gate를 가지고 있음 => gate는 cell state를 보호 & 제어
- LSTM블록의 중간에 있는 더하기(+)
  => 일반적인 RNNs의 유닛은 곱하기로만 이루어져있는데, LSTM은 피드백을 더하기로 연결
  => sigmoid를 곱하다 보니 생기는 그라디언트 소실 문제가 없는 것

<< Step >>

1. 삭제 게이트 (forget gate layer)

   - 셀 상태로부터 어떤 정보를 버릴것인가 => 시그모이드레이어에 의해서 결정
     -> gate(forget gate layer)
     => 0과 1사이의 값을 이전시점의 셀 상태값이 sigmoid layer 전달 = 컴포넌트가 얼마나 정보를 전달 해야하는지 파악

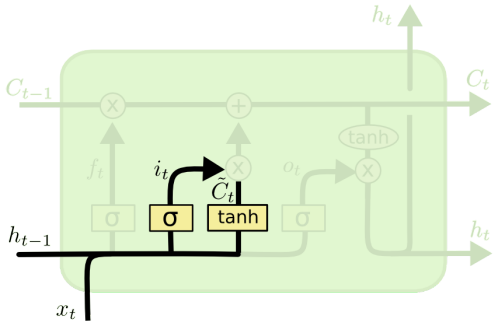
     => 1 : 넘지마
     => 0 : 다 넘겨



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

2. 입력 게이트 (input gate layer)

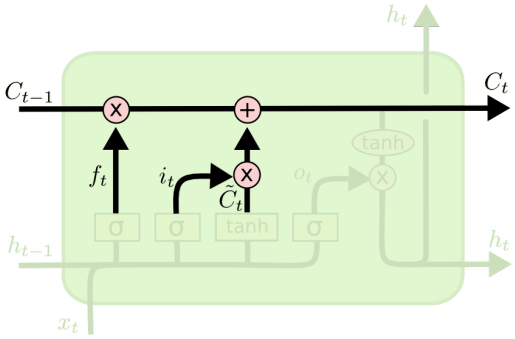
- 새로운 정보중 어떤것을 cell state에 저장 할지 선택 -> sigmoid layer = "input gate layer" -> sigmoid layer가 정한다 어떤값을 업데이트 할지 => tanh layer가 새로운 후보 값 => 벡터를 생성 그리고 cell state에 더한다 => 이게 업데이트 할 재료



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

3. 셀 상태: 장기 상태 (cell update)

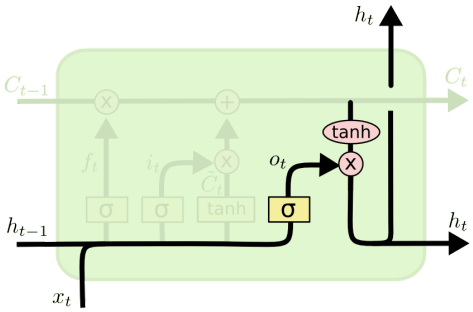
- 이전 시점의 셀상태 값을 업데이트 해서 새로운 셀상태를 만들것 => 전 단계에서 어떤값을 업데이트 할지 정해짐 삭제 게이트의 출력 값 첫단계에서 삭제한것을 진짜 삭제하고 첫단계에서 구한 두값을 곱한다. 그리고 더한다_ 아래 그림 참조



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

4. 출력 게이트와 은닉 상태 : 단기 상태 (Output gate layer)

- 무엇을 아웃풋으로 보낼것인가 ?
  => 시그모이드 레이어에 인풋 데이터를 태워서 셀 상태의 어느부분을 아웃풋으로 보낼지 정할것
  그리고 셀 상태에 하이퍼 볼릭 탄젠트에 태워 -1 , 1사이 값을 받은 후 시그모이드 게이트의 아웃풋에 곱한다.



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

식

h_{t-1} :(과거) 은닉상태

x_t : (현) x값

C_t : (현재) 셀 상태 => LSTM에서는 장기상태라고 부르기도 함

C_{t-1}: (과거) 셀 상태(cell state)

$\tilde{c}_t$: 새로운 후보 값 벡터

f_t : forget data

i_t : Input data

o_t : Output data

σ : Sigmoid 함수

$tanh()$: 하이퍼볼릭탄젠트함수

W_{xi}, W_{xg}, W_{xf}, W_{xo} : x_t와 함께 사용되는 4개의 가중치

W_{hi}, W_{hg}, W_{hf}, W_{ho} : h_{t-1}과 함께 each gate에서 사용되는 4개의 가중치

b_i, b_g, b_f, b_o : each gate 사용되는 4개의 편향

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$g_t = tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g)$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$
$$C_t = f_t \circ C_{t-1} + i_t \circ g_t$$
$$O_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$
$$h_t = O_t \circ tanh(c_t)$$

코드 구조 .. ㅎㅎ

1. 임포트
2. 데이터 셋 로드
   - 데이터 전처리
3. 데이터 처리
4. 필요한 변수 및 추가적인 변수 선언
   - EX) steps, Batch size, classes, hidden layer ...
5. 가설 설정 (모델생성)
   - LSTM
   - output, state
   - pred

   - 형변환 (Casting)
   - 텐서의 구조를 변경
   - 각 sequence마다 input을 연산

6. 비용& 손실함수
   - 옵티마이저
7. 세션생성및 세션 변수 초기화

8. 학습진행
9. 저장

In [3]:

```
# 임포트
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import numpy as np
```

```
C:\Users\JerryKim\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow
\python\framework\dtypes.py:516: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be understoo
d as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\JerryKim\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow
\python\framework\dtypes.py:517: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be understoo
d as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\JerryKim\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow
\python\framework\dtypes.py:518: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be understoo
d as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\JerryKim\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow
\python\framework\dtypes.py:519: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be understoo
d as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\JerryKim\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow
\python\framework\dtypes.py:520: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be understoo
d as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\JerryKim\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow
\python\framework\dtypes.py:525: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be understoo
d as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\JerryKim\AppData\Local\Continuum\anaconda3\lib\site-packages\h5py\__init
__.py:36: FutureWarning: Conversion of the second argument of issubdtype from `fl
oat` to `np.floating` is deprecated. In future, it will be treated as `np.float64
== np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
C:\Users\JerryKim\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorboard
\compat\tensorflow_stub\dtypes.py:541: FutureWarning: Passing (type, 1) or '1typ
e' as a synonym of type is deprecated; in a future version of numpy, it will be u
nderstood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\JerryKim\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorboard
\compat\tensorflow_stub\dtypes.py:542: FutureWarning: Passing (type, 1) or '1typ
e' as a synonym of type is deprecated; in a future version of numpy, it will be u
nderstood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\JerryKim\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorboard
\compat\tensorflow_stub\dtypes.py:543: FutureWarning: Passing (type, 1) or '1typ
e' as a synonym of type is deprecated; in a future version of numpy, it will be u
nderstood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\JerryKim\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorboard
\compat\tensorflow_stub\dtypes.py:544: FutureWarning: Passing (type, 1) or '1typ
e' as a synonym of type is deprecated; in a future version of numpy, it will be u
```

```
nderstood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:₩Users₩JerryKim₩AppData₩Local₩Continuum₩anaconda3₩lib₩site-packages₩tensorboard
₩compat₩tensorflow_stub₩dtypes.py:545: FutureWarning: Passing (type, 1) or '1typ
e' as a synonym of type is deprecated; in a future version of numpy, it will be u
nderstood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:₩Users₩JerryKim₩AppData₩Local₩Continuum₩anaconda3₩lib₩site-packages₩tensorboard
₩compat₩tensorflow_stub₩dtypes.py:550: FutureWarning: Passing (type, 1) or '1typ
e' as a synonym of type is deprecated; in a future version of numpy, it will be u
nderstood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

팁 : LSTM 하이퍼파라미터 정하기

LSTM의 하이퍼파라미터를 정하는 팁을 몇 가지 적어놓았으니 참고하십시오.

과적합(overfitting)이 일어나는지를 계속 모니터링하십시오. 과적합은 신경망이 학습 데이터를 보고 패턴을 인식하는 것이 아니라 그냥 데이터를 외워버리는 것인데 이렇게 되면 처음 보는 데이터가 왔을 때 제대로 결과를 내지 못합니다. 학습 과정에서 규제(regularization)가 필요할 수도 있습니다. l1-규제, l2-규제, 드롭아웃을 고려해보십시오. 학습엔 사용하지 않는 시험 데이터(test set)를 별도로 마련해두십시오. 신경망이 커질수록 더 많고 복잡한 패턴을 인식할 수 있습니다. 그렇지만 신경망의 크기를 키우면 신경망의 파라미터의 수가 늘어나게 되고 결과적으로 과적합이 일어날 수 있습니다. 예를 들어 10,000개의 데이터로 수백만개의 파라미터를 학습하는 것은 무리입니다. 데이터는 많으면 많을수록 좋습니다. 같은 데이터로 여러 번 학습을 시켜야합니다. 조기 종료(early stopping)을 활용하십시오. 검증 데이터(validation set)를 대상으로 얼마나 성능이 나오는지 확인하면서 언제 학습을 끝낼 것인지를 정하십시오. 학습 속도(learning rate)를 잘 설정하는 것은 중요합니다. DL4J의 ui를 써서 학습 속도를 조절해보십시오. 이 그래프를 참고하십시오. 다른 문제가 없다면 레이어는 많을수록 좋습니다. LSTM에서는 하이퍼탄젠트보다 softsign함수를 사용해보십시오. 속도도 더 빠르고 그라디언트가 평평해지는 문제(그라디언트 소실)도 덜 발생합니다. RMSProp, AdaGrad, Nesterove’s momentum을 적용해보십시오. Nesterove’s momentum에서 시작해서 다른 방법을 적용해보십시오. 회귀 작업에서는 데이터를 꼭 정규화하십시오. 정말 중요합니다. 또, 평균제곱오차(MSE)를 목적 함수로 하고 출력층의 활성함수(activation function)은 y=x(identity function)을 사용하십시오. (역자 주: 회귀 작업이라도 출력값의 범위를 [0,1]로 제한할 수 있다면 binary cross-entropy를 목적 함수로 사용하고 출력층의 활성함수는 sigmoid를 사용하십시오.)