딥러닝을 활용한 자연어 처리(2)

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

08

APPLICATION

ARTIFICIAL INTELLIGENCE

Notice

THINKING

생각해보기

✓ LSTM이란 무엇일까요?

학습목표

GOALS

1 BPTT의 문제점에 대해 설명할 수 있다.

2 Truncated BPTT 모델에 대해 설명할 수 있다.

3 LSTM의 기본 구조에 대해 배우고 설명할 수 있다.

4 LSTM의 작동 원리와 다양한 변형에 대해 설명할 수 있다.

1 BPTT의 문제점

2 LSTM이란?

3 실습

CONTENTS

학습내용

APPLICATION

01 BPTT의 문제점

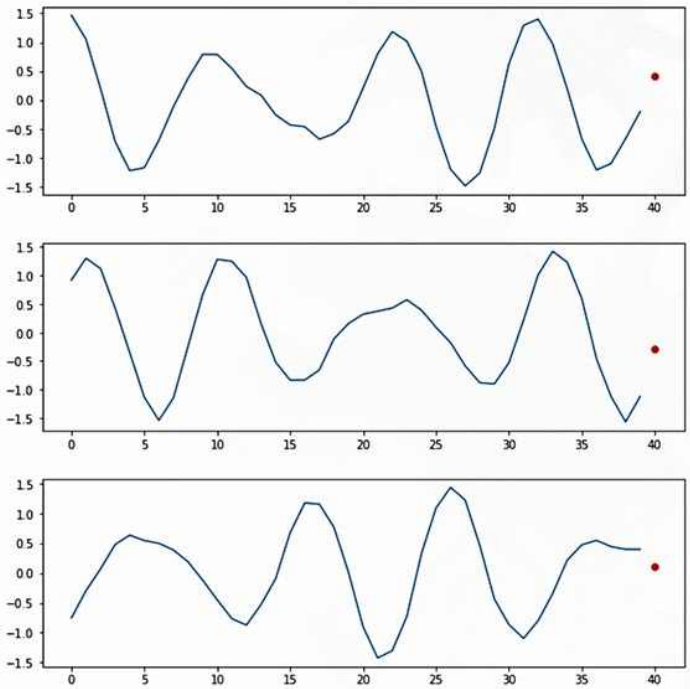
01 데이터셋 확인하기

Toy 데이터셋 생성 및 시각화 코드

In [2]:
```
def generate_timeseries(n_steps=50):
    m_x = np.random.uniform(0,10)
    xs = np.linspace(0, 5, n_steps+1)
    ys = np.array([0.5*np.sin(2*np.pi*(x+m_x))+ np.cos(3*np.pi/2*(x+m_x/4))
                   + np.random.uniform(-0.1,0.1) for x in xs])
    return ys[:-1],ys[-1]
```

In [3]:
```
for _ in range(3):
    xs, ys = generate_timeseries(n_steps=40)
    timesteps = np.arange(len(xs)+1)
    plt.figure(figsize=(10,3))
    plt.plot(timesteps[:-1],xs)
    plt.scatter(timesteps[-1],ys,c='r')
    plt.show()
```

01 데이터셋 확인하기



출처: 퍼블릭에이아이(www.publicai.co.kr)

02 Orthogonal Initialization

BPTT는 아래 수식과 같이 전파됨

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial h^{(t)}} = \frac{\partial a^{(t)}}{\partial h^{(t-1)}} = \frac{\partial L}{\partial h^{(t)}} tanh'(a^{(t)}) W_{hh}$$

$tanh'(a^{(t)})$를 1이라 가정하면,

$$\frac{\partial L}{\partial h_{(t-1)}} \approx \frac{\partial L}{\partial h^{(t)}} W_{hh}$$

$$\frac{\partial L}{\partial h_{(t-n)}} \approx \frac{\partial L}{\partial h^{(t)}} (W_{hh})^n$$

- 위 수식과 같이 N번의 Time step을 진행하게 되면, w_{hh}가 N번 반복하는 행렬곱 연산이 진행됨
- 초기 w_{hh}의 값에 따라 오차가 매우 크게 증폭될 수도 있고, 매우 적게 수렴될 수도 있음

예시 가중치 행렬

```
In [4]: 1 weight = np.random.normal(0,1,100).reshape(10,10)
        2 weight
```

```
Out[4]: array([[ 0.03666672,  1.14462696, -1.55855757,  0.25725379,  0.44290247,
                 1.26518414, -1.09747364,  1.69151188, -0.3968234 , -0.94287618],
               [ 0.83478675,  0.73430764, -2.00259076, -0.59689709, -1.27100364,
                 0.06933196,  1.21235025, -0.14346242,  0.63731411,  0.56899636],
               [ 2.23138081,  0.48392561, -0.63027686,  0.27665831, -0.92449675,
                -1.36810241, -1.57418077,  1.46998694, -0.55244965,  1.97029936],
               [-1.72754611, -0.41435054,  2.68611976,  1.06339836,  1.41128119,
                 1.74977474, -0.48256734, -0.29362928, -0.26223485,  0.182659  ],
               [-1.66942256, -0.14890885, -1.7337741 ,  0.09497742,  1.20939228,
                -0.39216772, -0.75343946, -0.27748395, -1.10705429, -2.03275895],
               [-0.46950145, -0.12333366,  2.34394509, -0.5598477 ,  0.89208253,
                 0.37619777, -1.37937798,  0.27719313,  1.24715435, -1.37888094],
               [ 0.99659677,  0.41758172, -0.52570925, -0.92064117,  0.58563962,
                -2.17556473, -0.66147625,  0.47084351,  2.43858949,  0.62481204],
               [-1.18522447, -0.44005893,  0.77904987, -1.63628225,  0.24606948,
                 1.88711752,  0.70078068, -0.1999333 ,  0.66620615, -0.07557188],
               [ 0.39754849, -0.3423693 , -1.65937283, -1.9490424 ,  0.52290445,
                 1.22188239, -0.98648085,  1.47108205,  0.14381132,  0.14657481],
               [ 0.43921183, -0.15593859, -0.05207265,  0.96379082,  0.84222914,
                -1.06585957,  0.92050749, -0.3247324 , -0.2580772 , -0.42342514]])
```

02 Orthogonal Initialization

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

가중치 행렬의 크기 : 보통 L2 norm으로 계산

```
In [5]: 1 np.sqrt(np.sum(weight**2))
```

```
Out[5]: 10.956754373466538
```

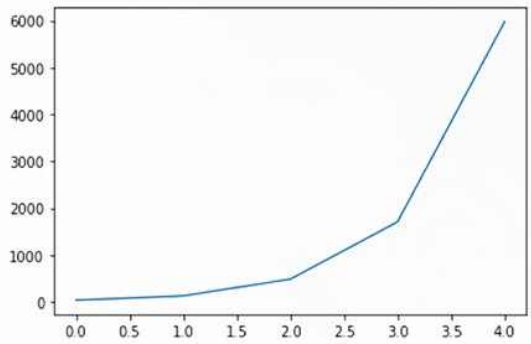
위의 수식을 Numpy에서는 아래와 같이 메소드를 제공합니다.

```
In [6]: 1 np.linalg.norm(weight)
```

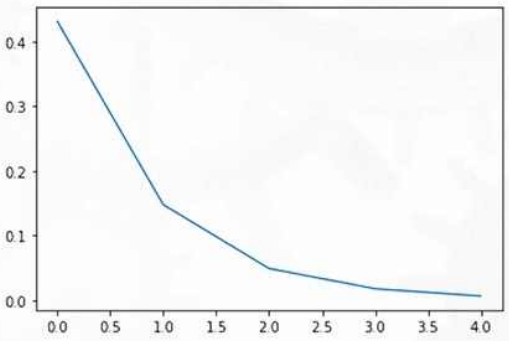
```
Out[6]: 10.956754373466538
```

연산이 거듭됨에 따른 가중치의 변화

```
w = weight.copy()
w_norms = []
for i in range(5):
    w = np.matmul(weight,w)
    w_norms.append(np.linalg.norm(w))

plt.plot(w_norms)
plt.show()
```




```
weight = np.random.normal(0,0.1,100).reshape(10,10)

w = weight.copy()
w_norms = []
for i in range(5):
    w = np.matmul(weight,w)
    w_norms.append(np.linalg.norm(w))

plt.plot(w_norms)
plt.show()
```



02 Orthogonal Initialization

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Orthogonal Initialization

- SVD(Sinlge Value Decomposition)를 통해 가중치 행렬의 각 행이 모두 수직이 되도록 만드는 방법

$$S = U\Sigma V$$

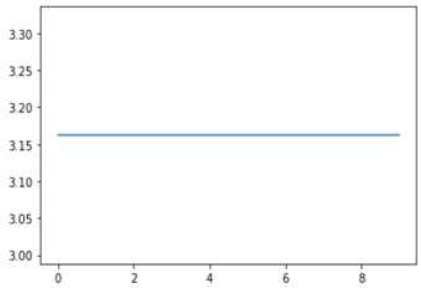
U matrix가 바로 직교 행렬

U matrix의 각 행은 Eigen Vector로 구성되어 있음

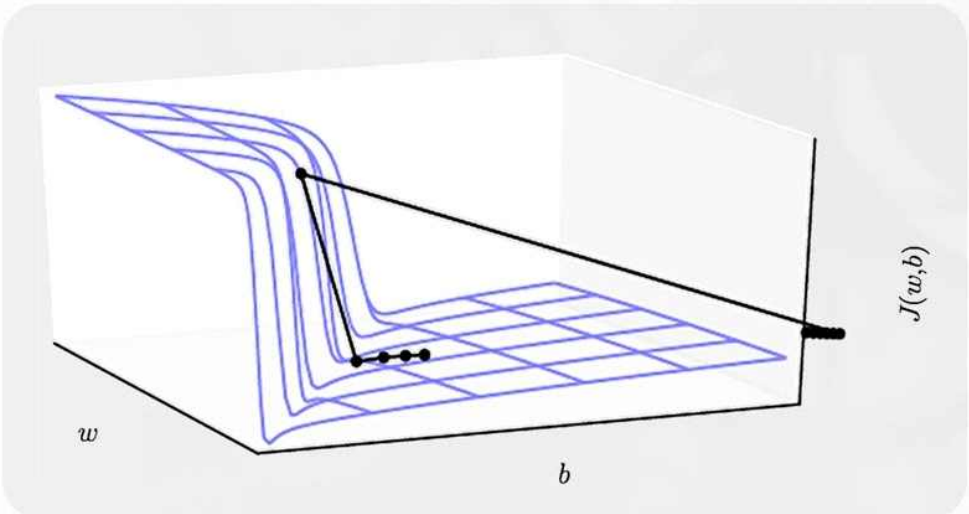
Orthogonal Initialization 수행 시 가중치 추세

In [9]:
```
weight = np.random.normal(0,1.,100).reshape(10,10) #
orthogonal, _, _ = np.linalg.svd(weight, full_matrices=False)
```

In [10]:
```
orth = orthogonal.copy()
orth_norms = []
for i in range(10):
    orth = np.matmul(orthogonal,orth)
    orth_norms.append(np.linalg.norm(orth))

plt.plot(orth_norms)
plt.show()
```



Gradient Clipping이란?



Gradient exploding 시의 문제 재현 - 모델 구성하기

In [12]:
```
K.clear_session()

# 모델 구성하기
n_inputs = 1
n_steps = 50
n_neurons = 200
n_outputs = n_inputs

inputs = Input(shape=(n_steps,n_inputs))
rec_init = tf.keras.initializers.RandomNormal(0,1.)
hidden = SimpleRNN(n_neurons,
                   recurrent_initializer=rec_init)(inputs)
output = Dense(1)(hidden)

model = Model(inputs,output)

model.compile(loss='mse', optimizer=Adam(lr=1e-4))
```

03 Gradient Clipping

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Gradient exploding 시의 문제 재현 - 모델 학습하기

In [13]:
```
def timeseries_generator(n_steps=50, batch_size=32):
    while True:
        batch_xs, batch_ys = [], []
        for _ in range(batch_size):
            x, y = generate_timeseries(n_steps)
            batch_xs.append(x[:,np.newaxis])
            batch_ys.append(y[np.newaxis])
        yield np.stack(batch_xs), np.stack(batch_ys)

batch_size = 16
train_gen = timeseries_generator(n_steps, batch_size)

hist = model.fit_generator(train_gen,
                           steps_per_epoch=100,
                           epochs=1)
```

```
WARNING:tensorflow:From <ipython-input-13-8390d579baba>:15: Model.fit_generator (from tensorflow.python.keras.engine.
training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
100/100 [==============================] - 3s 34ms/step - loss: nan
```

Gradient Clipping

- Gradient의 최대 크기를 제한하고, 만약 최대치를 넘게 되면 Gradient의 크기를 재조정하는 방법

$$\hat{g} = \frac{\partial \epsilon}{\partial \theta}$$

$$\begin{cases} \hat{g} \leftarrow \frac{threshold}{||\hat{g}||}\hat{g}, & \text{if } |\hat{g}| \geq threshold \\ \hat{g} \leftarrow \hat{g}, & \text{if } |\hat{g}| < threshold \end{cases}$$

03 Gradient Clipping

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Gradient Clipping - 모델 구성하기

In [14]:

```
K.clear_session()
n_inputs = 1
n_steps = 50
n_neurons = 200
n_outputs = n_inputs

inputs = Input(shape=(n_steps,n_inputs))
rec_init = tf.keras.initializers.RandomNormal(0,1.)
hidden = SimpleRNN(n_neurons,
                   recurrent_initializer=rec_init)(inputs)
output = Dense(1)(hidden)

model = Model(inputs,output)

# Optimizer에 clipnorm=0.1을 지정하는 것으로
# gradient clipping 적용가능
model.compile(loss=tf.losses.mean_squared_error,
              optimizer=Adam(lr=1e-4, clipnorm=.1))
```

Gradient Clipping - 모델 학습하기

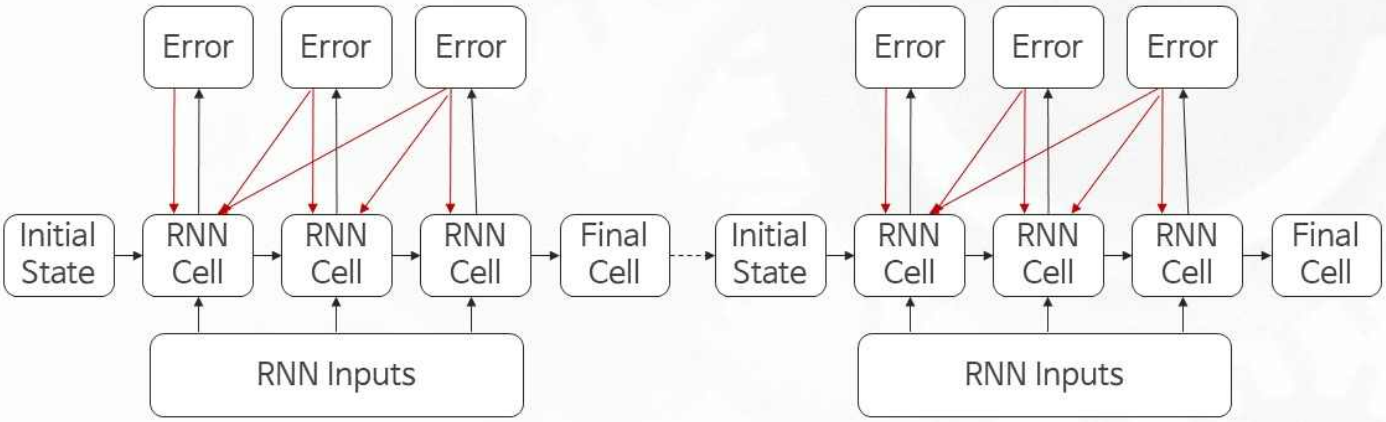
In [15]:
```
batch_size = 16
train_gen = timeseries_generator(n_steps, batch_size)

hist = model.fit_generator(train_gen,
                           steps_per_epoch=100,
                           epochs=10)
```

```
Epoch 1/10
100/100 [==============================] - 3s 35ms/step - loss: 2.4639
Epoch 2/10
100/100 [==============================] - 3s 34ms/step - loss: 2.2776
Epoch 3/10
100/100 [==============================] - 3s 33ms/step - loss: 2.1678
Epoch 4/10
100/100 [==============================] - 3s 35ms/step - loss: 2.1198
Epoch 5/10
100/100 [==============================] - 3s 34ms/step - loss: 2.0176
Epoch 6/10
100/100 [==============================] - 3s 33ms/step - loss: 1.9450
Epoch 7/10
100/100 [==============================] - 4s 35ms/step - loss: 1.8178
Epoch 8/10
100/100 [==============================] - 3s 33ms/step - loss: 1.7278
Epoch 9/10
100/100 [==============================] - 3s 33ms/step - loss: 1.5797
Epoch 10/10
100/100 [==============================] - 4s 35ms/step - loss: 1.6647
```


Truncated BPTT

- 학습 시 Error의 전파를 특정 구간별로 자르는 방법
- 본질적으로 BPTT 과정에서 문제가 발생되는 부분은 긴 시계열 데이터를 다루는 경우
- 긴 시계열 데이터의 경우 Gradient를 계산하기 위해 각 Time step 별로 값을 저장하기 때문에 메모리 이슈와 Gradient 이슈가 존재

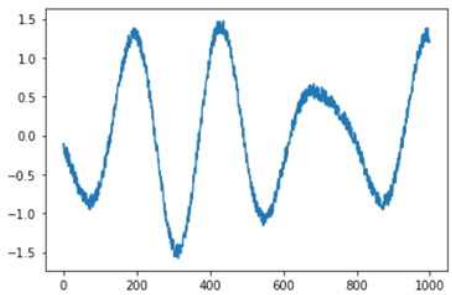
Truncated BPTT



데이터 확인하기

In [16]:
```
xs, _ = generate_timeseries(n_steps=1000)
```

In [17]:
```
plt.plot(xs)
plt.show()
```



모델 구성하기

In [18]:
```
K.clear_session()

# 모델 구성하기
n_inputs = 1
n_steps = 200
n_neurons = 200
batch_size = 16
n_outputs = n_inputs

inputs = Input(shape=(None,n_inputs), batch_size=batch_size)
hidden = SimpleRNN(n_neurons, stateful=True)(inputs)
output = Dense(1)(hidden)

model = Model(inputs, output)

model.compile(loss='mse',
              optimizer=Adam(lr=1e-4))
```


Stateful 을 위한 Generator 구성하기

In [19]:
```
total_ts = []
final_ys = []
for _ in range(batch_size):
    timeseries, y = generate_timeseries(n_steps=1000)
    total_ts.append(timeseries)
    final_ys.append(y)

total_ts = np.stack(total_ts)
final_ys = np.stack(final_ys)[:,np.newaxis]
```

In [20]:
```
print("total_ts의 형태 : ",total_ts.shape)
print("final_ys의 형태 : ",final_ys.shape)
```

```
total_ts의 형태 :  (16, 1000)
final_ys의 형태 :  (16, 1)
```

```
train_xs = (total_ts.reshape((batch_size,-1,n_steps)))
print("train_xs의 형태 :", train_xs.shape)
```

```
train_xs의 형태 : (16, 5, 200)
```

```
train_ys = np.concatenate([train_xs[:,1:,0], final_ys],axis=1)
print("train_ys의 형태 :", train_ys.shape)
```

```
train_ys의 형태 : (16, 5)
```

```
train_xs = train_xs.transpose(1,0,2)[...,np.newaxis]
train_ys = train_ys.transpose(1,0)[...,np.newaxis]
print("train_xs의 형태 :", train_xs.shape)
print("train_ys의 형태 :", train_ys.shape)
```

```
train_xs의 형태 : (5, 16, 200, 1)
train_ys의 형태 : (5, 16, 1)
```

Stateful 을 위한 Generator 구성하기

In [24]:
```
def generate_stateful_sequences(batch_size=16,
                                truncated_steps=50,
                                n_steps=1000):
    total_ts = []
    final_ys = []
    for _ in range(batch_size):
        timeseries, y = generate_timeseries(n_steps)
        total_ts.append(timeseries)
        final_ys.append(y)

    total_ts = np.stack(total_ts)
    final_ys = np.stack(final_ys)[:,np.newaxis]

    train_xs = (total_ts.reshape((batch_size,-1,truncated_steps)))
    train_ys = np.concatenate([train_xs[:,1:,0], final_ys],axis=1)

    train_xs = train_xs.transpose(1,0,2)[...,np.newaxis]
    train_ys = train_ys.transpose(1,0)[...,np.newaxis]
    return train_xs, train_ys
```


Truncated BPTT 적용하기

In [25]:
```
n_epochs = 30

for i in range(n_epochs):
    train_xs, train_ys = generate_stateful_sequences()

    model.reset_states() # state값을 0벡터로 초기화 시켜줌
    for batch_xs, batch_ys in zip(train_xs, train_ys):
        model.train_on_batch(batch_xs, batch_ys)

    if i % 10 == 0:
        model.reset_states() # state값을 0벡터로 초기화 시켜줌
        loss = model.evaluate(batch_xs, batch_ys, verbose=0)
        print("[{:2d} epoch] loss : {:.3f}".format(i, loss))
```

```
[ 0 epoch] loss : 0.005
[10 epoch] loss : 0.010
[20 epoch] loss : 0.005
```

APPLICATION

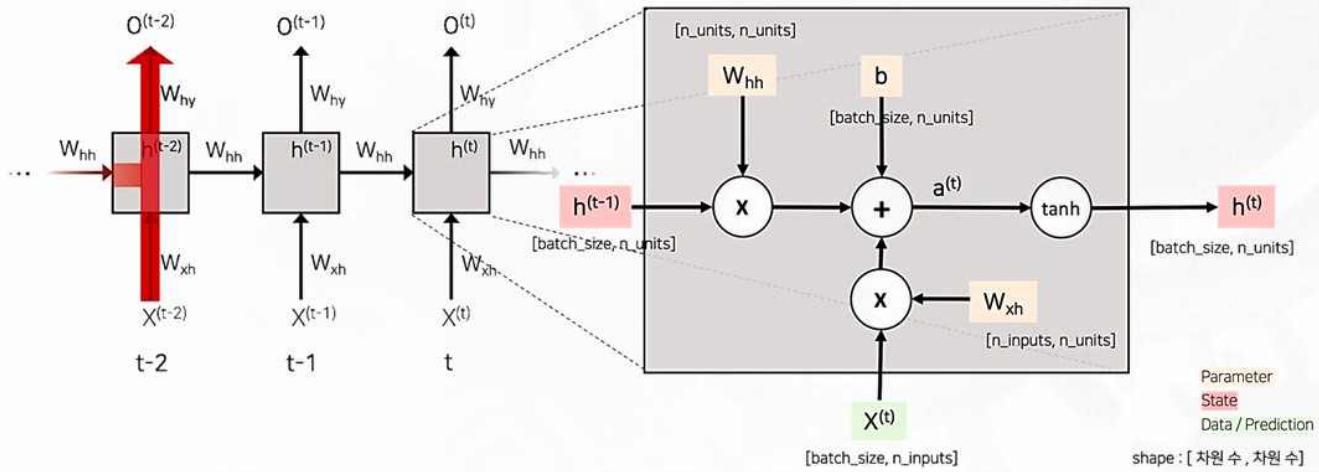
02 LSTM이란?

LSTM(Long Short-Term Memory)

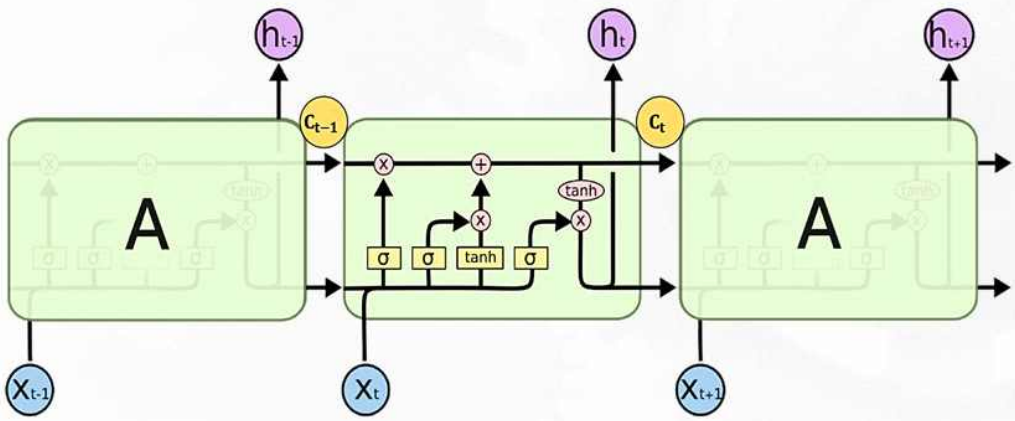
- RNN의 hidden state에 cell-state를 추가한 구조
- Input과 output 사이에 긴 time step이 존재할 때 학습에 어려움이 있는 BPTT의 이슈를 개선하고자 고안된 모델

RNN의 Cell을 개선하여 해결하고자 함

기존의 RNN Cell

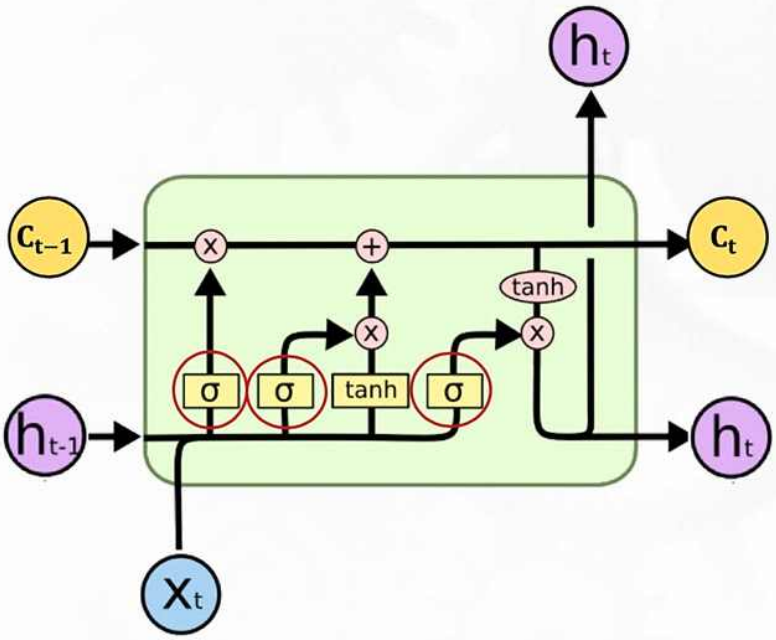


출처: 퍼블릭에이아이(www.publicai.co.kr)




```
In [3]:
class LSTMCell(Layer):
    def __init__(self, n_units, **kwargs):
        self.n_units = n_units
        # 두 개의 State (C,H)를 가지기 때문에
        # self.state_size는 (n_units, n_units)가 됩니다.
        self.state_size = (n_units, n_units)
        super(LSTMCell, self).__init__(**kwargs)
```

출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

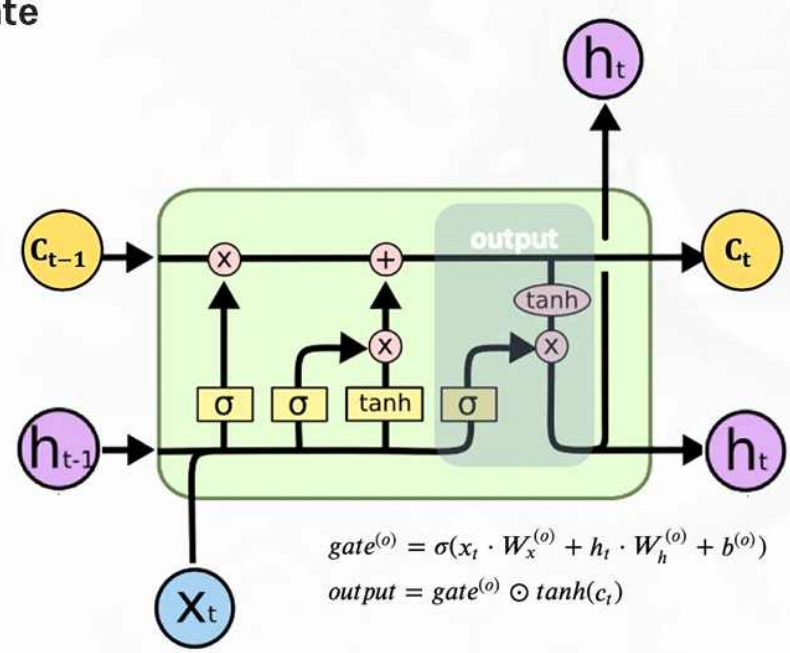


출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

03 LSTM의 다양한 Gate

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Output Gate



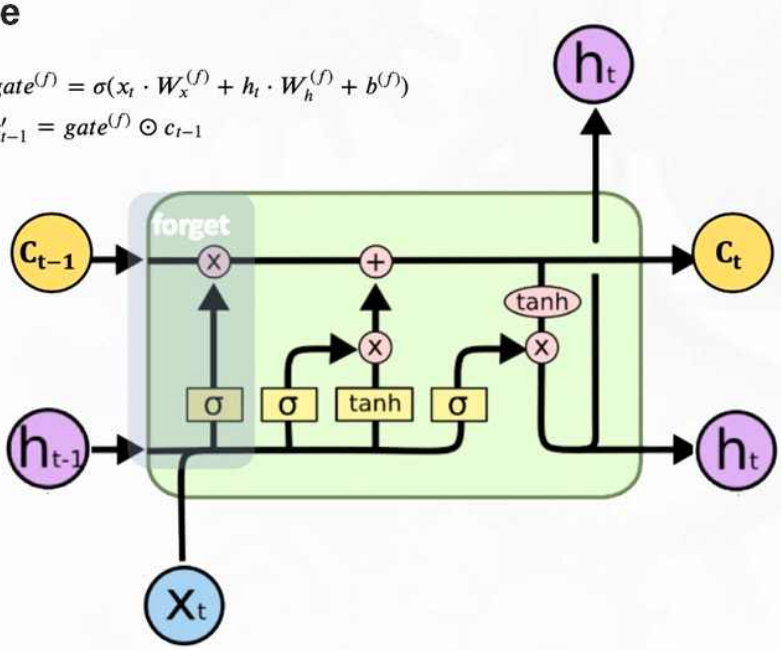
$$gate^{(o)} = \sigma(x_t \cdot W_x^{(o)} + h_t \cdot W_h^{(o)} + b^{(o)})$$

$$output = gate^{(o)} \odot tanh(c_t)$$

출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Forget Gate

$$gate^{(f)} = \sigma(x_t \cdot W_x^{(f)} + h_t \cdot W_h^{(f)} + b^{(f)})$$
$$c'_{t-1} = gate^{(f)} \odot c_{t-1}$$



출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

$$gate^{(f)} = \sigma(x_t \cdot W_x^{(f)} + h_t \cdot W_h^{(f)} + b^{(f)})$$
$$gate^{(o)} = \sigma(x_t \cdot W_x^{(o)} + h_t \cdot W_h^{(o)} + b^{(o)})$$
$$gate^{(i)} = \sigma(x_t \cdot W_x^{(i)} + h_t \cdot W_h^{(i)} + b^{(i)})$$
$$update = tanh(x_t \cdot W_x^{(u)} + h_t \cdot W_h^{(u)} + b^{(u)})$$
$$- - - - - - - - - - -$$
$$c_t = gate^{(f)} \odot c_{t-1} + gate^{(i)} \odot update$$
$$h_t = gate^{(o)} \odot tanh(c_t)$$

$$x_t \cdot W_x^{(f)} + h_t \cdot W_h^{(f)} + b^{(f)}$$
$$x_t \cdot W_x^{(o)} + h_t \cdot W_h^{(o)} + b^{(o)}$$
$$x_t \cdot W_x^{(i)} + h_t \cdot W_h^{(i)} + b^{(i)}$$
$$x_t \cdot W_x^{(u)} + h_t \cdot W_h^{(u)} + b^{(u)}$$

- 이 식을 정리하면 아래처럼 구현이 됩니다.

$$x_t \cdot [W_x^{(f)}, W_x^{(o)}, W_x^{(i)}, W_x^{(u)}] + h_t \cdot [W_h^{(f)}, W_h^{(o)}, W_h^{(i)}, W_h^{(u)}] + [b^{(f)}, b^{(o)}, b^{(i)}, b^{(u)}]$$
$$= x_t \cdot W_x + h_t \cdot W_h + b$$

05 LSTM Cell 구성

In [3]:

```
class LSTMCell(Layer):
    def __init__(self, n_units, **kwargs):
        self.n_units = n_units
        self.state_size = (n_units, n_units)
        super(LSTMCell, self).__init__(**kwargs)

    def build(self, input_shape):
        self.wx = self.add_weight("weight_x",
                                  shape=(input_shape[-1],self.n_units*4),
                                  initializer='glorot_uniform')
        self.wh = self.add_weight('weight_h',
                                  shape=(self.n_units,self.n_units*4),
                                  initializer='orthogonal')
        self.b = self.add_weight('bias',
                                 shape=(self.n_units*4,),
                                 initializer='zeros')
        super(LSTMCell, self).build(input_shape)

    def call(self, x, states):
        h, c = states
        # 행렬곱 연산
        z = tf.dot(x, self.wx) + tf.dot(h, self.wh) + self.b

        # forget에 관련된 처리들
        forget_fate = tf.sigmoid(z[:,:self.n_units])
        # update에 관련된 처리들
        input_gate = tf.sigmoid(z[:,self.n_units:self.n_units*2])
        update = tf.tanh(z[:,self.n_units*2:self.n_units*3])
        new_c = forget_gate * c + update * input_gate
        # output에 관련된 처리들
        output_gate = tf.sigmoid(z[:,self.n_units*3:])
        new_h = output_gate * tf.tanh(new_c)

        return output, [new_h, new_c]
```

06 Keras를 활용한 LSTM

In [4]:

```
from tensorflow.keras.layers import LSTMCell, RNN, Input, Dense
from tensorflow.keras.models import Model

K.clear_session()
n_steps = 30
n_inputs = 50
n_hiddens = 100
n_outputs = 3

inputs = Input(shape=(n_steps,n_inputs))
hidden = RNN(LSTMCell(n_hiddens))(inputs)
output = Dense(n_outputs,activation='softmax')(hidden)

model = Model(inputs,output)

model.summary()
```

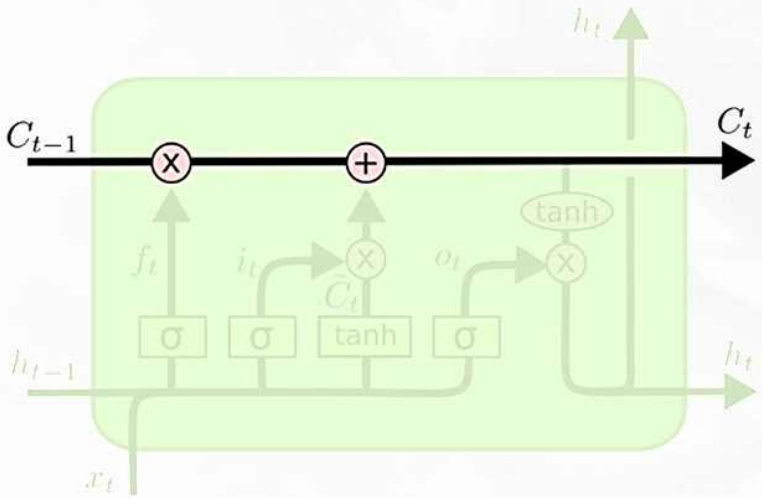
```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 30, 50)]          0
_________________________________________________________________
rnn (RNN)                    (None, 100)               60400
_________________________________________________________________
dense (Dense)                (None, 3)                 303
=================================================================
Total params: 60,703
Trainable params: 60,703
Non-trainable params: 0
```

06 Keras를 활용한 LSTM

In [5]:
```
from tensorflow.keras.layers import LSTM

K.clear_session()
n_steps = 30
n_inputs = 50
n_hiddens = 100
n_outputs = 3

inputs = Input(shape=(n_steps,n_inputs))
hidden = LSTM(n_hiddens)(inputs)
output = Dense(n_outputs,activation='softmax')(hidden)

model = Model(inputs,output)

model.summary()
```

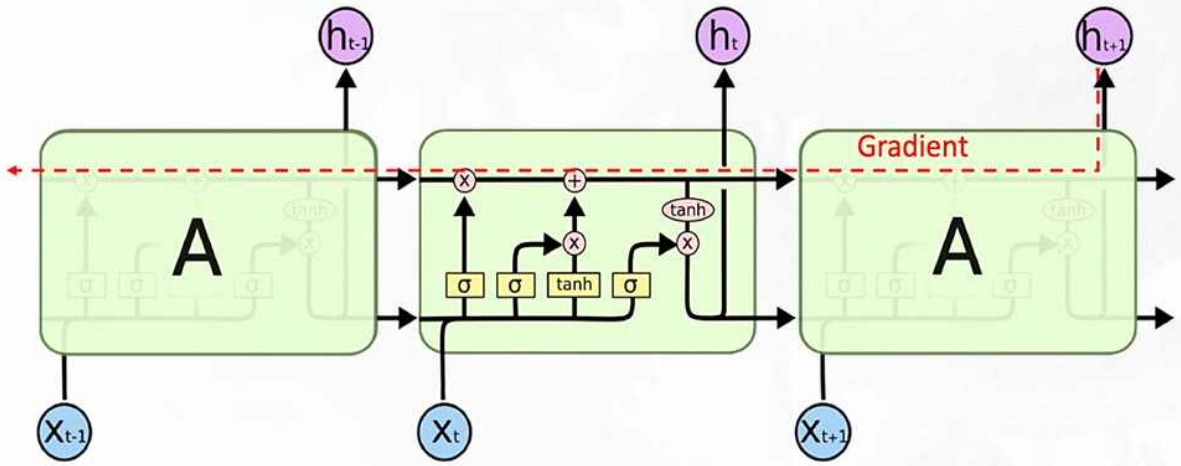
```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 30, 50)]          0
_________________________________________________________________
lstm (LSTM)                  (None, 100)               60400
_________________________________________________________________
dense (Dense)                (None, 3)                 303
=================================================================
Total params: 60,703
Trainable params: 60,703
Non-trainable params: 0
```


07 LSTM의 순전파



출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

08 LSTM의 역전파



출처: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

09 LSTM의 메모리와 연산량

LSTM

RNN에 비해 훨씬 더 많은 Weights를 가지고 있음

→ 3개의 gate와 하나의 update로 인해, 총 4배의 weight

추가적으로 3번의 sigmoid 연산과 1번의 tanh 연산이 더해지기 때문에 Cell에서 처리하는 데에 좀 더 많은 시간을 필요로 함

APPLICATION

03 실습

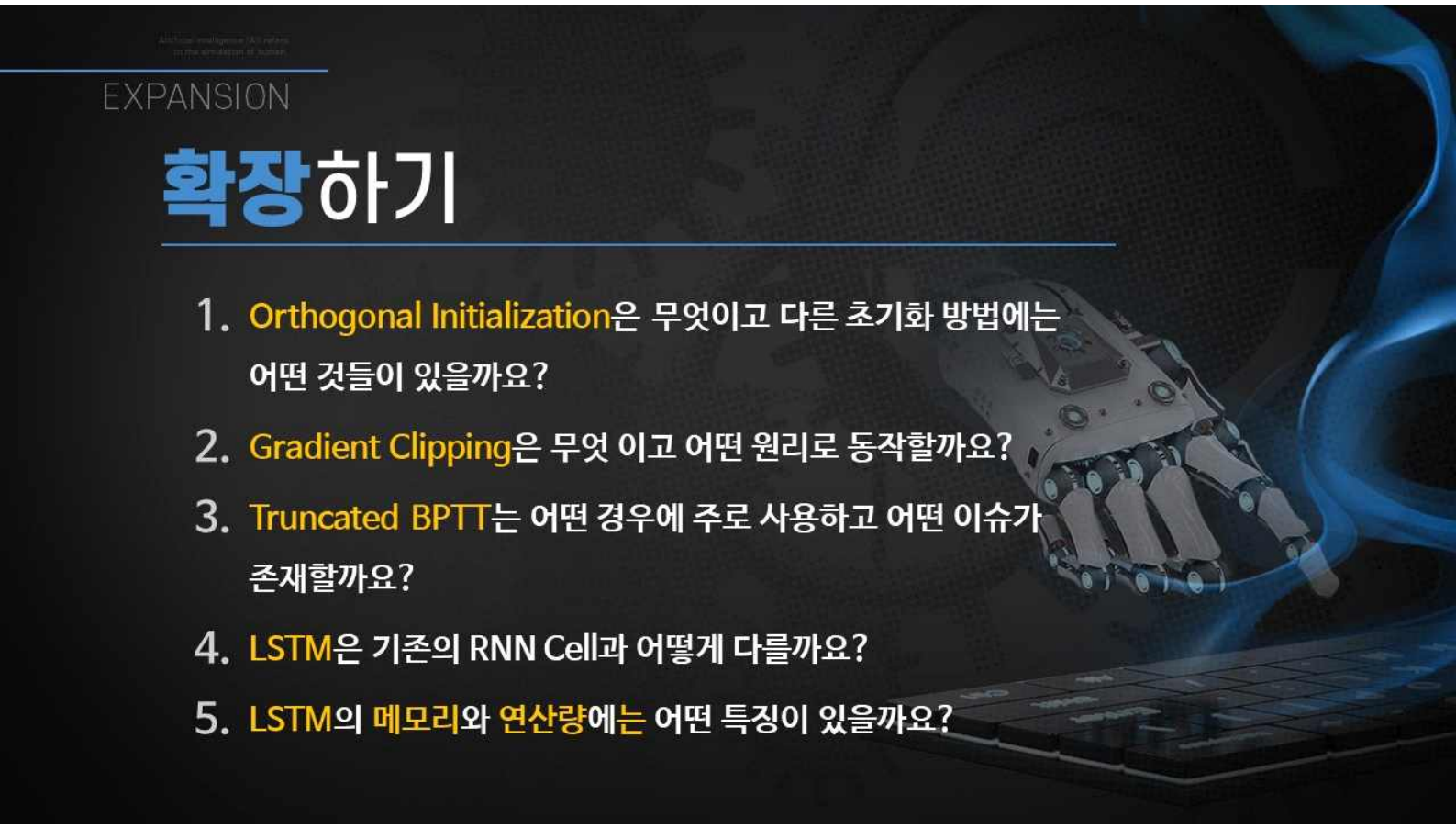
SUMMARY

학습정리

- BPTT의 문제를 해결하기 위한 Orthogonal Initialization
- Recurrent weight의 초기화 방법 Gradient Clipping
- Error의 전파를 특정 구간별로 자르는 Truncated BPTT
- LSTM(Long Short-Term Memory)

EXPANSION

확장하기



1. Orthogonal Initialization은 무엇이고 다른 초기화 방법에는 어떤 것들이 있을까요?
2. Gradient Clipping은 무엇 이고 어떤 원리로 동작할까요?
3. Truncated BPTT는 어떤 경우에 주로 사용하고 어떤 이슈가 존재할까요?
4. LSTM은 기존의 RNN Cell과 어떻게 다를까요?
5. LSTM의 메모리와 연산량에는 어떤 특징이 있을까요?

참고 문헌

REFERENCE

◆ 참고 사이트
- 용어들에 대한 정의 : https://ko.wikipedia.org/wiki.
- Understandig LSTM Networks : http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- 퍼블릭에이아이(www.publicai.co.kr)

◆ 참고 서적
- 사이토 고키, 『밑바닥 부터 시작하는 딥러닝 2』, 한빛미디어, 2019

서체 출처 : 에스코어드림체-㈜에스코어, 나눔글꼴체-㈜네이버