Deep learning

Keras: Deep Learning library for TensorFlow

Ha-Jin Yu, Dept. of Computer Science, University of Seoul
서울시립대학교 컴퓨터과학부 유하진

2019.

HJYU@UOS.AC.KR

서울시립대학교
UNIVERSITY OF SEOUL

Deep learning 도구

- Python : 쉽고 간편한 프로그래밍 언어
- Numpy : Package for scientific computing
- TensorFlow
- Pytorch
- Keras: Deep Learning library for TensorFlow

https://www.tensorflow.org/
http://keras.io

Keras

- Modular neural networks library
- Python, TensorFlow 기반
- Enable fast experimentation.
- Convolutional networks 와 recurrent networks 모두 지원
- 다양한 connectivity schemes (multi-input and multi-output training).
- Runs seamlessly on cpu and gpu.

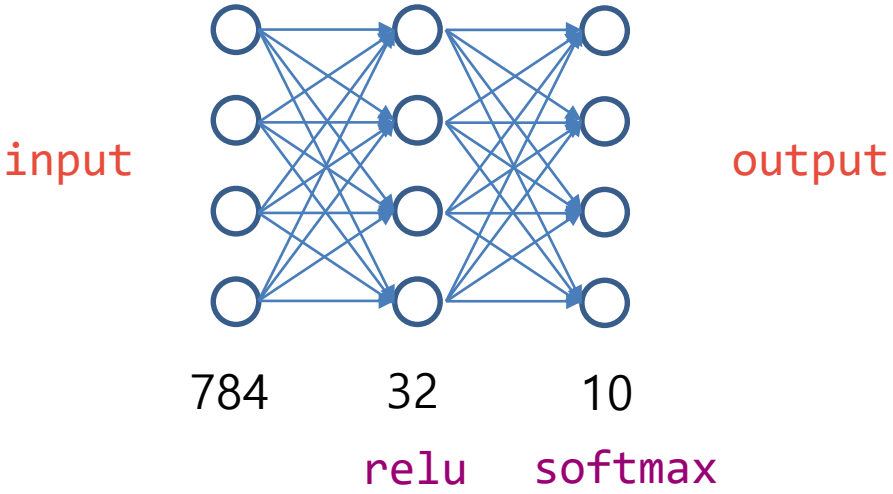
Keras

- Model = sequence or a graph of modules
- Modules
  - Neural layers
  - Cost functions
  - Optimizers
  - Initialization schemes
  - Activation functions
  - Regularization schemes

Keras 시작 -- Sequential model

- Sequential model : layer 들의 나열

```
from keras.models import Sequential
from keras.layers import Dense, Activation
model = Sequential([       # Regular fully connected NN layer
    Dense(32, input_dim=784),    # input 784, output 32
    Activation('relu'),
    Dense(10),                   # input 32, output 10
    Activation('softmax'),
])
```



Dense layer

- Regular fully connected NN layer.

```
keras.layers.core.Dense(output_dim, init='glorot_uniform',
activation='linear', weights=None, W_regularizer=None,
b_regularizer=None, activity_regularizer=None, W_constraint=None,
b_constraint=None, bias=True, input_dim=None)
```

- Example

```
# as first layer in a sequential model:
model = Sequential()
model.add(Dense(32, input_dim=16))
# now the model will take as input arrays of shape (*, 16)
# and output arrays of shape (*, 32)
# this is equivalent to the above:
model = Sequential()
model.add(Dense(32, input_shape=(16,)))
model.add(Activation('relu'))
# after the first layer, you don't need to specify
# the size of the input anymore:
model.add(Dense(32))
```

Dense layer -- Arguments

- **output_dim**: int > 0.
- **init**:
  - initialization function for the weights 또는 Tensorflow function
  - weights argument 가 없을 경우만 유효.
- **activation**: 없으면 "linear" activation: a(x) = x
- **weights**:
  - list of Numpy arrays to set as initial weights.
  - weights (input_dim, output_dim)
  - biases (output_dim,)
- **W_regularizer**: eg. L1 or L2 regularization
- **W_constraint**: eg. maxnorm, nonneg
- **bias**: whether to include a bias
  (i.e. make the layer affine rather than linear).

Recurrent layer - Abstract base class for recurrent layers

```
keras.layers.recurrent.Recurrent(weights=None, return_sequences=False,
go_backwards=False, stateful=False, unroll=False, consume_less='cpu',
input_dim=None, input_length=None)
```

→ Children classes LSTM, GRU and SimpleRNN.

Arguments

- weights: list of Numpy arrays to set as initial weights. The list should have 3 elements, of shapes: [(input_dim, output_dim), (output_dim, output_dim), (output_dim,)].
- return_sequences: Boolean. Whether to return the last output in the output sequence, or the full sequence.
- go_backwards: Boolean (default False).
  True이면 입력을 뒤 방향으로 처리.

Arguments

- stateful: Boolean (default False). True: 각 batch의 마지막 state가 다음 batch의 초기 state로 사용됨.
- unroll: Boolean (default False). True: 메모리 사용 ↑. 속도 ↑
- consume_less: "cpu", "mem", "gpu"(LSTM/GRU only) 중 선택 .
  - "cpu": 적은 수의 큰 행렬의 곱 연산을 수행하여 메모리를 많이 사용하고 CPU를 적게 사용.
  - "mem": 많은 수의 작은 행렬의 곱 연산을 수행하여 메모리를 적게 사용하고 CPU를 많이 사용. GPU에서 유리.
  - "gpu" (LSTM/GRU only): input gate, forget gate, output gate를 하나의 행렬로 합하여 GPU에서 유리하도록.
- input_dim: dimensionality of the input (integer).
  첫번째 layer 에서 필요. (input_shape 로 대체 가능)

LSTM - Long-Short Term Memory unit

```
keras.layers.recurrent.LSTM(output_dim, init='glorot_uniform',
inner_init='orthogonal', forget_bias_init='one', activation='tanh',
inner_activation='hard_sigmoid', W_regularizer=None,
U_regularizer=None, b_regularizer=None, dropout_W=0.0, dropout_U=0.0)
```

Arguments

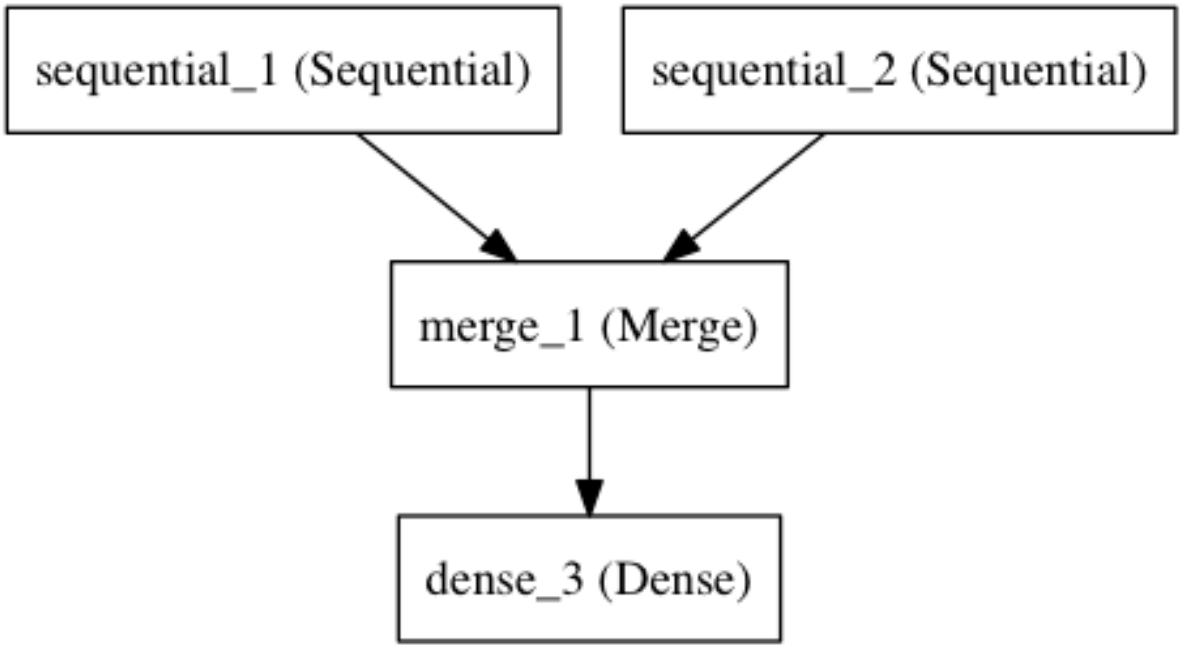
- **output_dim**: dimension of the internal projections and the final output.
- **init**: weight initialization function.
- **inner_init**: initialization function of the inner cells.
- **forget_bias_init**: recommend initializing with ones.
- **activation**: activation function.
- **inner_activation**: activation function for the inner cells.
- **W_regularizer**: input weights matrices.
- **U_regularizer**: recurrent weights matrices.
- **b_regularizer**: bias.
- **dropout_W**: [0 ,1]. input gates.
- **dropout_U**: [0 ,1]. recurrent connections.

The Merge layer

- Multiple Sequential instances can be merged into a single output via a Merge layer.
- ex) a model with two separate input branches getting merged:

```
from keras.layers import Merge

left_branch = Sequential()
left_branch.add(Dense(32, input_dim=784))

right_branch = Sequential()
right_branch.add(Dense(32, input_dim=784))
merged = Merge([left_branch, right_branch], mode='concat')

final_model = Sequential()
final_model.add(merged)
final_model.add(Dense(10, activation='softmax'))
```

The Merge layer



Trainning two-branch model

```
final_model.compile
      (optimizer='rmsprop', loss='categorical_crossentropy')

final_model.fit([input_data_1, input_data_2], targets)
# we pass one data array per model input
```

Pre-defined modes of the Merge layer

- sum (default): element-wise sum
- concat: tensor concatenation. You can specify the concatenation axis via the argument concat_axis.
- mul: element-wise multiplication
- ave: tensor average
- dot: dot product. You can specify which axes to reduce along via the argument dot_axes.
- cos: cosine proximity between vectors in 2D tensors.

Compilation - compile method

- Configure the learning process
- Arguments:
  - An optimizer.
    - The string identifier of an existing optimizer (such as rmsprop or adagrad), or an instance of the optimizer class.
  - A loss function.
    - The objective that the model will try to minimize.
    - The string identifier of an existing loss function (such as categorical_crossentropy or mse), or an objective function.
  - A list of metrics.
    - For classification problem : metrics=['accuracy'].
    - Or a custom metric function.

Training - fit function

- Keras models are trained on Numpy arrays of input data and labels.

```
# for a single-input model with 2 classes (binary):
model = Sequential()
model.add(Dense(1, input_dim=784, activation='softmax'))
model.compile(optimizer='rmsprop',
      loss='binary_crossentropy',
      metrics=['accuracy'])

# generate dummy data
import numpy as np
data = np.random.random((1000, 784))
labels = np.random.randint(2, size=(1000, 1))

# train the model, iterating on the data in batches
# of 32 samples
model.fit(data, labels, epochs=10, batch_size=32)
```

array([1, 0, 0, 0, 1, 1, 0, 0, 1, 0])

```
# for a multi-input model with 10 classes:
left_branch = Sequential()
left_branch.add(Dense(32, input_dim=784))
right_branch = Sequential()
right_branch.add(Dense(32, input_dim=784))
merged = Merge([left_branch, right_branch], mode='concat')
model = Sequential()
model.add(merged)
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='rmsprop',
          loss='categorical_crossentropy',
          metrics=['accuracy'])
# generate dummy data
import numpy as np
from keras.utils.np_utils import to_categorical
data_1 = np.random.random((1000, 784))
data_2 = np.random.random((1000, 784))
# these are integers between 0 and 9
labels = np.random.randint(10, size=(1000, 1))
# we convert the labels to a binary matrix of size (1000, 10)
# for use with categorical_crossentropy
labels = to_categorical(labels, 10)

# train the model
# note that we are passing a list of Numpy arrays as training data
# since the model has 2 inputs
model.fit([data_1, data_2], labels, epochs=10, batch_size=32)
```

Examples

- CIFAR10 small images classification: Convolutional Neural Network (CNN) with realtime data augmentation
- IMDB movie review sentiment classification: LSTM over sequences of words
- Reuters newswires topic classification: Multilayer Perceptron (MLP)
- MNIST handwritten digits classification: MLP & CNN
- Character-level text generation with LSTM

keras를 이용하여 AND, OR, XOR 실험

```
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD
import numpy as np
hiddendim = 2
inputdim = 2
outdim = 1
batsize = 4
nepoch = 100000
x_train = np.array([[0.,0.], [0.,1.], [1.,0.], [1.,1.]])
y_train = np.array([0,1,1,0])
x_test = np.array([[0.,0.], [0.,1.], [1.,0.], [1.,1.]])
y_test = np.array([0,1,1,0])
```

```
model = Sequential()
model.add(Dense(hiddendim, input_dim=inputdim,
        kernel_initializer='uniform', activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
        optimizer='rmsprop', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=1000, batch_size=batsize)
score = model.evaluate(x_test, y_test, batch_size=batsize)
print( score)
output = model.predict_classes(x_test, batch_size=4,
        verbose=1)
print( output)
output = model.predict_proba(x_test, batch_size=4,
        verbose=1)
print( output)
```

MNIST-Dense

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test,  y_test, verbose=2)
```

MNIST-CNN-1

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models

(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))

# 픽셀 값을 0~1 사이로 정규화.
train_images, test_images = train_images / 255.0, test_images / 255.0

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

MNIST-CNN-2

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.summary()

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=5)

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

예제: LSTM으로 계산기 구현

- LSTM을 사용한 산술 연산 구현 예제
- LSTM에 일정한 규칙을 가지고 생성된 수식들을 학습시킴으로써 수식이 포함하고 있는 산술 연산을 구현
- 본 예제에서는 덧셈을 포함하고 있는 수식들을 LSTM으로 학습

Ex) 학습시키고자 하는 수식이 '13+24=37'인 경우,

' 13+24=' 입력, '3' 출력

' 13+24=3' 입력, '7' 출력

'13+24=37' 입력, '.' 출력

(점은 수식의 끝을 의미)

LSTM으로 계산기 구현

```
import numpy as np
# 학습할 수식의 최대 길이 설정
# 두자릿수 + 두자릿수의 경우 온점을 포함 최대 10개 문자가 올 수 있음
max_len = 10
# 미리 테스트에 사용할 수식들을 정의해
# 학습 데이터에서 제외
test_str_list = ['10+10=20.', '99+99=198.', '13+24=37.']
# 두자릿수 덧셈에서 발생 가능한 모든 수식을 생성
# (테스트 데이터 제외)
train_str_list = []
for i in range(1, 100):
        for j in range(1,100):
                a = str(i)
                b = str(j)
                c = str(i+j)
                sentence = a +'+'+b+'='+c+'.'
```

```
                    # 테스트에 사용할 수식들은 제외
                    for test in test_str_list:
                            if sentence in test:
                                    continue
                    # 최대 길이에서 부족한 만큼 공백으로 padding
                    pad = ' ' * (max_len - len(sentence))
                    sentence = pad + sentence
                    train_str_list.append(sentence + '₩n')
# LSTM의 효율적인 학습을 위해 생성한 학습 수식들의 순서를 임의로 변경
np.random.shuffle(train_str_list)
# text 파일의 형태로 생성한 학습 수식들을 저장
f= open('expressions_train.txt', 'w')
for train in train_str_list:
        f.write(train)
f.close()
```

문자열 학습

```
from __future__ import print_function
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.utils.data_utils import get_file
import numpy as np
import random
import sys
from keras.models import model_from_json
# 학습시킬 수식의 최대 길이 정의
maxlen = 10
# 미리 저장해둔 학습 수식 로드
fn ='expressions_train.txt'
text = open(fn).read().lower()
print('corpus length:', len(text))
# 학습 데이터에 포함돼 있는 모든 문자 집합 생성
chars =set(text)
print('total chars:', len(chars))
```

```
# index -> 문자, 문자 -> index 관계를 나타내는 dictionary를 각각 생성
char_indices =dict((c, i) for i, c in enumerate(chars))
indices_char = dict((i, c) for i, c in enumerate(chars))
# 학습 수식들을 사용해 LSTM에 학습시킬
# 입력 문장(sentences)과 출력 문자(next_chars) 쌍 생성
# Ex) ' 13+24=' 입력, '3' 출력
sentences =[]
next_chars = []
for line in open(fn):
        line = line.strip()
        exp_ready = False
        for i in range(len(line)):
                if exp_ready:
                        sentences.append(line[: i])
                        next_chars.append(line[i])
                # 등호 기호 이전의 문자는 모두 입력 받기 때문에 예측할 필요 없음
                if line[i] is '=':
                        exp_ready = True
print('nb sequences:', len(sentences))
```

```
# 학습시킨 LSTM 모델에 수식을 입력해 결과값을 확인하는 함수
def test_model(model):
# 테스트에 사용할 수식들을 미리 정의
# '9+19='과 '1+1='은 학습 데이터에 포함돼 있는 수식
# '10+10='과 '99+99=', '13+24='은 학습 데이터 생성시 미리 제외시켜둔 수식
#
test_list = ["9+19=","1+1=","10+10=","99+99=","13+24="]
```

```
for test in test_list:
    sentence = test
    generated = ''
    generated += sentence
    sys.stdout.write(generated)
    for i in range(maxlen):
        # 테스트 sentence를 LSTM에 입력하기 위해 vectorization
        x = np.zeros((1, maxlen, len(chars)))
        for t, char in enumerate(sentence):
        # 정의한 최대 길이에서 부족한 만큼 수식의 앞부분을 padding
            x[0,t+(maxlen - len(sentence)), char_indices[char]]=1.
        preds = model.predict(x, verbose=0)[0]
        next_index = np.argmax(preds)
        next_char = indices_char[next_index]
        # 예측한 문자를 기존 문장에 추가하여 다시 예측에 사용
        generated += next_char
        sentence = sentence + next_char
        sys.stdout.write(next_char)
        sys.stdout.flush()
```

```
                    # 수식의 끝을 나타내는 온점이 나올 때까지 예측 반복
                    if next_char is '.':
                        break
            print()
# 앞에서 생성한 입력 문장, 출력 문자 쌍을 vectorization
X= np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)
y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
for i, sentence in enumerate(sentences):
    for t, char in enumerate(sentence):
        if char == ' ':
            continue
        # 정의한 최대 길이에서 부족한 만큼 수식의 앞부분을 padding
        X[i, t + (maxlen - len(sentence)), char_indices[char]] = 1
    y[i, char_indices[next_chars[i]]] = 1
```

57+91=148

57+91=

57+91=1

57+91=14

57+91=148

1

4

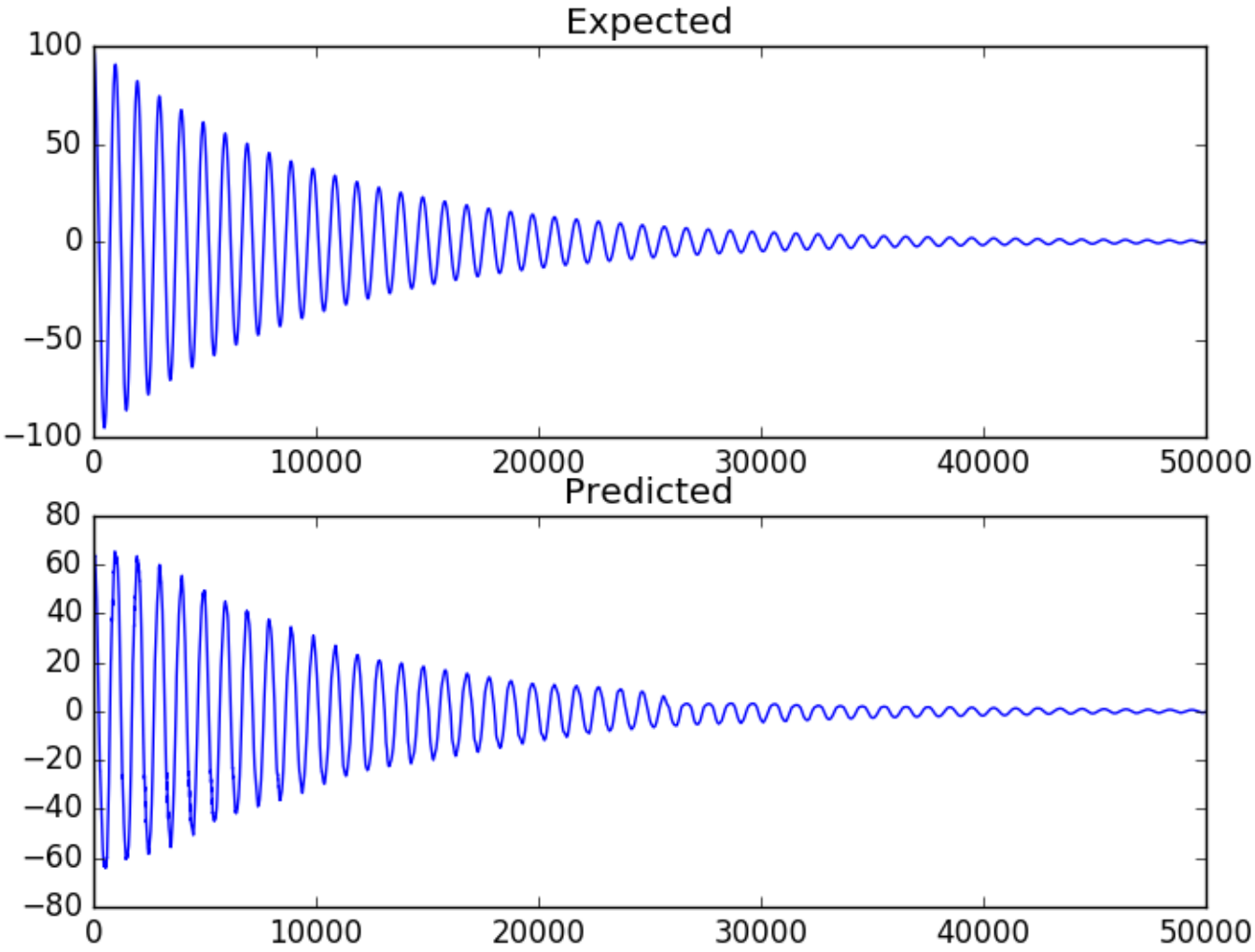
8

.

```
# LSTM 모델 구성
print('Build model...')
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape=(maxlen,
        len(chars))))
model.add(Dropout(0.2))
model.add(LSTM(128, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(len(chars)))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
model.reset_states()
```

```
# 60회 반복 학습 수행
for iteration in range(1, 60):
        print()
        print('-' * 50)
        print('Iteration', iteration)
        model.fit(X, y, batch_size=128, epochs=1)
        # 학습된 모델의 성능을 확인
        test_model(model)
# 정해진 횟수 만큼 학습을 마친 모델을 저장
json_string =model.to_json()
# LSTM 구성을 저장
open('my_LSTM_architecture.json', 'w').write(json_string)
# 학습한 weights 등의 파라미터들을 저장
model.save_weights('my_LSTM_weights.h5')
```

LSTM 예제: 신호 값 예측



LSTM 예제: 신호 값 예측

```
# Data: amplitude가 점점 감소하는 형태의 cosine 신호
import numpy as np
def gen_cosine_amp(amp=100, period=25, length=50000, k=0.0001):
    """Generates a cosine time series with the amplitude exponentially
    decreasing
    Arguments:
        amp: amplitude of the cosine function
        period: period of the cosine function
        k: exponential rate
    """
    cos = np.zeros((length, 1, 1))
    for i in range(length):
            cos[i, 0, 0] = amp * np.cos(i / (2 * np.pi * period))
            cos[i, 0, 0] = cos[i, 0, 0] * np.exp(-k * i)
    return cos
```

LSTM 예제: 신호 값 예측

필요한 module 및 매개변수 설정

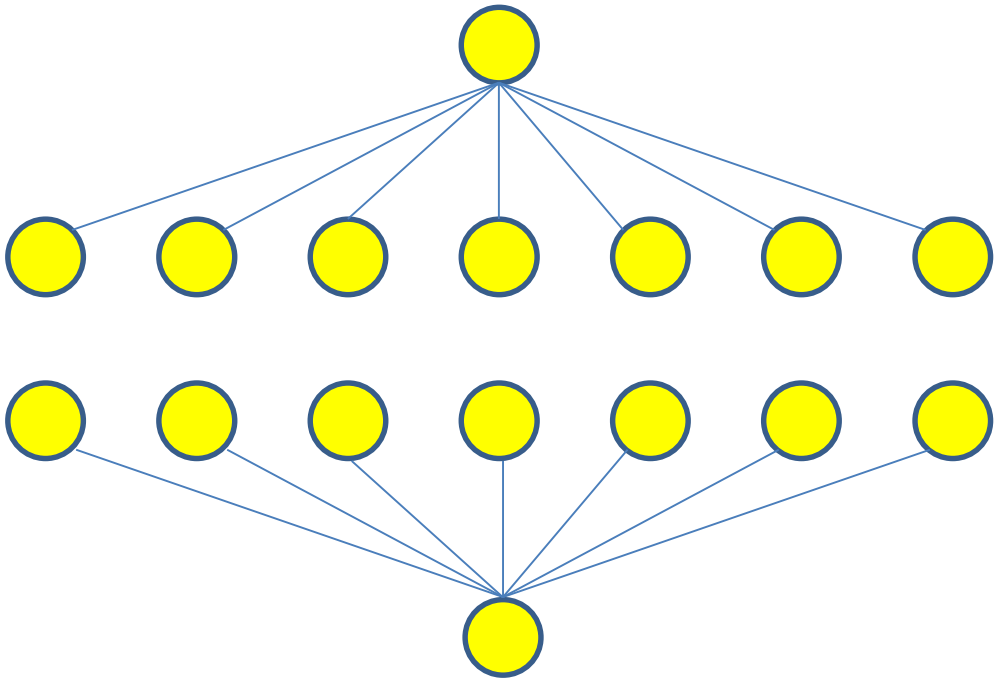
```
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, LSTM

tsteps = 1 # time step
batch_size = 25 # batch size
epochs = 25 # 학습 반복 횟수
lahead = 1 # 출력 data의 값을 구성하는 데 필요한 sample 수

cos = gen_cosine_amp() # Data 생성
# Output: 현재 sample 다음에 나올 예측값
# 현재 sample이 i번째 sample이라고 하면, 예측값은 [i+1번째 ~ i+lahead+1번째]
까지의 sample의 평균(lahead개 sample들의 평균)
expected_output = np.zeros((len(cos), 1))
for i in range(len(cos) - lahead):
        expected_output[i, 0] = np.mean(cos[i + 1:i + lahead + 1])
```

Model 생성

```
model = Sequential()
# LSTM layer 추가
model.add(LSTM(50, # hidden layer의 node 수
               batch_input_shape=(batch_size, tsteps, 1),
# 각 batch에 사용되는 input data의 형태 (각 batch 당 sample 수, time_step, 차원수)
# tsteps : 몇 개의 step을 보고 prediction 할 것인지
# stateful=True로 설정한 경우에는 batch_input_shape 매개변수를 사용해야하며,
# mod(len(data), batch_size) = 0을 만족해야함
               return_sequences=True,
# True: 출력 sequence 전체를 출력 (h0 ~ ht)
# False: 출력 sequence의 마지막 부분만 출력 (ht)
# 다음 layer에 RNN과 같은(e.g. LSTM) layer를 추가하고자 하는 경우, True로 설정
               stateful=True))
# 이전 단계의 batch에 사용된 sample들의 state가 현재 단계의 batch의 sample들의 초기 state로 사용됨
# 즉, 이전 단계의 batch에 해당하는 data subset과
# 현재 단계의 batch에 해당하는 data subset의 관계가 이어지는 경우 True로 설정
```

LSTM layer 및 Output layer추가

```
# LSTM layer 추가
model.add(LSTM(50, batch_input_shape=(batch_size, tsteps, 1),
               return_sequences=False, stateful=True))
# Output layer 추가
model.add(Dense(1))
model.compile(loss='mse', optimizer='rmsprop')
```



학습 및 평가

```
# epoch 횟수만큼 학습
for i in range(epochs):
        model.fit(cos, expected_output,
                batch_size=batch_size, verbose=1, epochs=1,
                shuffle=False)
# 본 예제에서는 sample의 순서가 유지되어야 하므로 shuffle을 수행하지 않음
        model.reset_states()
# 다음 epoch를 수행하기 위해 model의 cell state 초기화

# 값 예측
predicted_output = model.predict(cos, batch_size=batch_size)
```

결과 표시

```
print('Plotting Results')
plt.subplot(2, 1, 1)
plt.plot(expected_output)
plt.title('Expected')
plt.subplot(2, 1, 2)
plt.plot(predicted_output)
plt.title('Predicted')
plt.show()
```

이미지 파일 읽기

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
img = mpimg.imread('lena.png')
# RGB 각 채널에 접근
R_channel = img[:,:,0] #R 채널
G_channel = img[:,:,1] #G 채널
B_channel = img[:,:,2] #B 채널
# 이미지 내 특정 구역에 노이즈 삽입
# 특정 구역 정의
x_position = 200
x_length = 100
y_position = 200
y_length = 100
```

```
img_block = img[y_position:y_position+ y_length, x_position: x_position
                + x_length]
# 노이즈 데이터 생성
#
noise_level = 2.5 # 잡음 정도 설정
# 작은 값일수록 높은 잡음 정도를 의미
noise = np.random.rand(y_length, x_length, 3)
# 잡음 값을 임의로 생성한 뒤, 정규화
noise /= np.max(noise)
noise -= np.mean(noise)
noise /= noise_level
img_block += noise # 선택한 특정 구간에 잡음 삽입
img[y_position:y_position+ y_length,
          x_position: x_position + x_length] = img_block
plt.imshow(img)
plt.show()
```

엑셀 파일 읽기

```
# -*- coding: utf-8 -*-
import xlrd
from xlrd.sheet import ctype_text
fn = 'iris_data.xlsx'
# 엑셀 파일 로드
xl_workbook = xlrd.open_workbook(fn)
# 로드한 엑셀 파일에 포함돼 있는 sheet 확인
sheet_names = xl_workbook.sheet_names()
print('Sheet Names', sheet_names)
# sheet의 이름을 기준으로 해당 sheet에 접근
xl_sheet = xl_workbook.sheet_by_name(sheet_names[0])
```

```
# sheet의 index를 기준으로 해당 sheet에 접근
xl_sheet = xl_workbook.sheet_by_index(0)
print ('Sheet name: %s' % xl_sheet.name)
# 접근한 sheet 내에서 다시 접근하고자 하는 row의 index를 기준으로 접근
row = xl_sheet.row(0) #첫번째 row에 접근
# 첫번째 row에 저장돼 있는 값들을 출력
print('(Column #) type:value')
for idx, cell_obj in enumerate(row):
      cell_type_str = ctype_text.get(cell_obj.ctype, 'unknown type')
      print('(%s) %s %s' % (idx, cell_type_str, cell_obj.value))
# row index와 colum index를 활용해 접근한 sheet에 포함돼 있는 모든 cell에 접근
num_cols = xl_sheet.ncols
for row_idx in range(0, xl_sheet.nrows):
    print ('-'*40)
    print ('Row: %s' % row_idx)
    for col_idx in range(0, num_cols):
            cell_obj = xl_sheet.cell(row_idx, col_idx)
            print ('Column: [%s] cell_obj: [%s]' % (col_idx, cell_obj))
```