Machine Learning Tools & Libraries on Python

March 15, 2018

Introduction

This document provides a brief tutorial on mainstream machine learning tools and libraries using Python (mainly 3), including a short introduction, links to official documents, along with some tips simple examples of frequently used functions or routines for each tool or library. The main goal of this documents is to help the readers (1) deploy classical machine learning algorithms in a short period of time, and (2) accelerate developing new augmented machine learning algorithms.

The following tools & libraries are covered in this document:

NumPy, SciPy, scikit-learn, sklearn-crfsuite, Pandas, TensorFlow

The following tools & libraries are being updated by the writers:

(None)

The following tools & libraries are to be updated sometime in the future by the writers:

Matplotlib, PyTorch

Contents

1 NumPy **5**

1.1 Introduction to NumPy . . . 5

1.2 Frequently Used Routines of NumPy . . . 5

1.2.1 Array creation . . . 5

1.2.2 Array manipulation . . . 5

1.2.3 Miscellaneous . . . 5

2 SciPy **6**

2.1 Introduction to SciPy . . . 6

2.2 Frequently Used Routines of SciPy . . . 6

2.2.1 Integration . . . 6

2.2.2 Optimization . . . 6

2.2.3 Linear algebra . . . 7

2.2.4 File IO . . . 7

3 scikit-learn **7**

3.1 Introduction to scikit-learn . . . 7

3.2 Frequently Used Routines . . . 7

3.2.1 Logistic regression . . . 8

3.2.2 Regularized logistic regression . . . 8

3.2.3 Linear regression . . . 8

3.2.4 Regularized linear regression . . . 8

3.2.5 Linear discriminant analysis . . . 9

3.2.6 Quadratic discriminant analysis . . . 9

3.2.7 Support vector machine . . . 9

3.2.8 Principle component analysis . . . 9

3.2.9 Latent Dirichlet allocation . . . 10

4 sklearn-crfsuite **10**

4.1 Introduction to sklearn-crfsuite 10

4.2 Frequently Used Models 10

4.2.1 Conditional random field 10

5 Pandas **12**

5.1 Introduction to Pandas 12

5.2 Frequently Used Models 12

5.2.1 Statistical time series 12

6 TensorFlow **13**

6.1 Introduction to TensorFlow 13

6.2 Frequently Used Models 13

6.2.1 Conditional random field 13

6.2.2 Convolutional neural network 13

6.2.3 Recurrent neural network 14

6.2.4 Deep Q learning 16

Index

Conditional random field, 10, 13
Convolutional neural network, 13

Latent Dirichlet allocation, 10
Linear discriminant analysis, 9
Linear regression, 5, 8
Logistic regression, 8

Principle component analysis, 9

Quadratic discriminant analysis, 9

Recurrent neural network, 14
Reinforcement learning, 16

Statistical time series, 12
Support vector machine, 9

Time series, 12, 14

1 NumPy

1.1 Introduction to NumPy

NumPy is a library for Python, adding support for large, multi-dimensional arrays and matrices. NumPy also supports a large collection of high-level mathematical functions to operate on these arrays and matrices, including Fourier transform, decomposition, eigenvalue, etc. NumPy is not a library with advanced algorithms, yet it is the base for multiple advanced machine learning tools & libraries.

The more detailed information of NumPy can be found **here**.

1.2 Frequently Used Routines of NumPy

The complete document of NumPy API can be found **here**.

1.2.1 Array creation

```
>>> numpy.empty([2, 3], dtype=numpy.float64)
```
create a random 2x3 64-bit float matrix

```
>>> numpy.eye(3, 4, 2, dtype=numpy.float32)
```
create a 3x4 32-bit float matrix. The elements in the diagonal that is 2 rows/columns above the main diagonal are all 1's, while all the other elements are 0's.

```
>>> numpy.array([4, 5, 6], dtype=complex, ndmin=2)
```
create a complex tensor of $[[4+0j, 5+0j, 6+0j]]$ as a 2-d(1x3) matrix rather than a vector.

More array creation routines can be found **here**.

1.2.2 Array manipulation

```
>>> numpy.reshape(M, (3, 2))
```
reshape M into a 3x2 matrix.

```
>>> numpy.flip(M, 0)
```
flip M regarding the first (index 0) dimension (row by row).

More array manipulation routines can be found **here**.

1.2.3 Miscellaneous

```
>>> numpy.fft.fft(M, n=3, axis=2)
```
compute fast Fourier transform on the first 3 elements of M over the third (index 2) dimension.

More discrete Fourier transform routines can be found **here**.

```
>>> w, b, null, null = numpy.linalg.lstsq(X, y)
```
fit a linear regression model $y = \mathbf{w}^T\mathbf{x} + b$.

More linear algebra routines can be found **here**.

2 SciPy

2.1 Introduction to SciPy

SciPy is an open source Python library used for scientific computing and technical computing, containing modules for optimization, linear algebra, integration, interpolation, special functions, Fourier transform, signal and image processing, ordinary differential equation solvers and other tasks common in science and engineering. SciPy builds upon the NumPy array object and therefore relies on NumPy.

The more detailed information of SciPy can be found **here**.

2.2 Frequently Used Routines of SciPy

The complete document of SciPy API can be found **here**.

2.2.1 Integration

```
>>> y, null = scipy.integrate.quad(lambda x:numpy.sin(x**3), 0, 2*numpy.pi)
```
compute the following integration as y:

$$y = \int_0^{2\pi} \sin(x^3)$$

```
>>> from scipy.integrate import dblquad
>>> z, null = dblquad(lambda x, y:  x*y, 0, 1, lambda x:  0, lambda x:  1-x)
```
compute the following double integration as z:

$$z = \int_0^1 \int_0^{1-x} xydydx$$

2.2.2 Optimization

```
>>> import numpy as np
>>> from scipy.optimize import minimize
>>> def func(x, sign=1.0):
...    return sign*(2*x[0]*x[1] + 2*x[0] - x[0]**2 - 2*x[1]**2)
```
define the objective function $f(x, y)$ as:

$$f(x, y) = -x^2 + 2xy - 2y^2 + 2x$$

```
>>> def func_d(x, sign=1.0):
...    dfdx0 = sign*(-2*x[0] + 2*x[1] + 2)
...    dfdx1 = sign*(2*x[0] - 4*x[1])
...    return np.array([dfdx0, dfdx1])
```
define the total derivatives of $f(x, y)$.
```
>>> cons = ('type':  'eq',
...        'fun' :  lambda x:  np.array([x[0]**3 - x[1]]),
...        'jac' :  lambda x:  np.array([3.0*(x[0]**2.0), -1.0]),
...        'type':  'ineq',
...        'fun' :  lambda x:  np.array([x[1] - 1]),
```

```
...       'jac' :  lambda x:  np.array([0.0, 1.0]))
```
define the constraints as:

$$y - 1 \geq 0$$
$$x^3 - y = 0$$

along with their total derivatives.
`res = minimize(func, [4,6], args=(-1,), jac=func_d, constraints=cons, method='SLSQP')`
start the optimization "$\max f(x, y)$", which is equivalent to "$\min -1 \times f(x, y)$", from $(x, y) = (4, 6)$.

2.2.3 Linear algebra

```
>>> from scipy import linalg as l
>>> l.pinv(A)
```
find the inverse (or pseudo inverse for non-square matrices) of A.

```
>>> l.solve(A, b)
```
find the solution of $A\mathbf{x} = \mathbf{b}$.

```
>>> l.det(A)
```
find the determinant of A.

```
>>> eval, evec = l.eig(A)
```
find the eigenvalues `eval` and eigenvectors `evec` of A.

2.2.4 File IO

```
>>> import scipy.io as io
>>> a = loadmat('D:\matfile1')
>>> savemat('D:\matfile2', {'adata':a['data']})
```
load a MATLAB file "D:\matfile1.mat" and save its "data" tensor into a new MATLAB file "D:\matfile2.mat" as "adata".

```
>>> import scipy.io.wavfile as wv
>>> null, a = wv.read('D:\audio1.wav')
>>> wv.write('D:\audio2.wav', 44100, a)
```
load a raw wave file "D:\audio1.wav" and save as a new raw wave file "D:\audio2.wav" at a sample rate of 44.1kHz.

3 scikit-learn

3.1 Introduction to scikit-learn

Scikit-learn is a free software machine learning library for Python. It features various classification, regression and clustering algorithms including discriminant analysis, support vector machines, random forests, gradient boosting, k-means, DBSCAN, etc. Scikit-learn relies on NumPy and SciPy.

3.2 Frequently Used Routines

The complete document of scikit-learn API can be found **here**.

3.2.1 Logistic regression

```
>>> from sklearn.linear_model import Perceptron
>>> import sklearn.datasets as ds
>>> from sklearn.preprocessing import StandardScaler
>>> d = ds.load_digits()
>>> X, y = d.data, d.target
```
load a predefined dataset classifying digits (from 0 to 9);
```
>>> X = StandardScaler().fit_transform(X)
```
normalize x to x_{norm} as:

$$x_{norm} = \frac{x - \mu(x)}{\sigma(x)}$$

```
>>> y = (y > 4).astype(int)
```
divide the digits into two classes: small (from 0 to 4) and large (from 5 to 9);
```
>>> lr = Perceptron()
>>> lr.fit(X, y)
```
create a logistic regression classifier and start training;
```
>>> yp = lr.predict(X)
```
predict $\hat{y}$ as `yp`.

3.2.2 Regularized logistic regression

```
>>> lr = Perceptron(penalty='elasticnet', alpha=0.5)
```
set the regularization term as elastic net `elasticnet` or ridge `l2` or lasso `l1`, and set the coefficient of the regularization term as 0.5.

3.2.3 Linear regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression()
>>> lr.fit(X, y)
>>> yp = lr.predict(X)
```
create a linear regression classifier with X and **y**, and predict $\hat{y}$ as `yp`.

Other optional parameters:
`fit_intercept=False`: no bias for the classifier;
`normalize=True`: normalize the features before training.

3.2.4 Regularized linear regression

```
>>> from sklearn.linear_model import Ridge
>>> lr = Ridge(alpha=0.5)
```
create a ridge regression (linear regression with L2-regularization) classifier with the coefficient of the regularization term as 0.5.

```
>>> from sklearn.linear_model import Lasso
>>> lr = Lasso(alpha=0.5)
```
create a lasso regression (linear regression with L1-regularization) classifier with the coefficient of the regularization term as 0.5.

Other optional parameters:
`fit_intercept=False`: no bias for the classifier;
`normalize=True`: normalize the features before training.

3.2.5 Linear discriminant analysis

```
>>> from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
>>> lc = LinearDiscriminantAnalysis(priors=[0.7, 0.3])
>>> lc.fit(X, y)
>>> yp = lc.predict(X)
```
create a linear discriminant analysis classifier with X and $\mathbf{y}$, setting the priors as $p(y=0)=0.7$ and $p(y=1)=0.3$, and predict $\hat{y}$ as `yp`.

Other optional parameters:
`n_components=a`: reduce the number of features into a using singular vector decomposition.

3.2.6 Quadratic discriminant analysis

```
>>> from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
>>> qc = QuadraticDiscriminantAnalysis(priors=[0.7, 0.3])
>>> qc.fit(X, y)
>>> yp = qc.predict(X)
```
create a quadratic discriminant analysis classifier with X and $\mathbf{y}$, setting the priors as $p(y=0)=0.7$ and $p(y=1)=0.3$, and predict $\hat{y}$ as `yp`.

Other optional parameters:
`n_components=a`: reduce the number of features into a using singular vector decomposition.
`reg_param=b`: regularize the covariance matrices Σ_i into Σ_i' of all classes as:

$$\Sigma_i' = (1-b)\Sigma_i + bI \qquad \text{for all classes } i$$

3.2.7 Support vector machine

```
>>> from sklearn.svm import LinearSVC
>>> lc = LinearSVC()
>>> lc.fit(X, y)
>>> yp = lc.predict(X)
```
create a linear support vector machine classifier with X and $\mathbf{y}$, and predict $\hat{y}$ as `yp`.

Other optional parameters:
`penalty='l1'`: change the regularization term from $\frac{\mathbf{w}^T\mathbf{w}}{2}$ to $||\mathbf{w}||$;
`loss='squared_hinge'`: change the hinge loss term from $C\sum_{i=1}^{N}\xi_i$ to $\frac{C}{2}\sum_{i=1}^{N}\xi_i^2$;
`dual=False`: solve the primal problem of SVM rather than the dual;
`C=a`: set the coefficient of the hinge loss as a.

3.2.8 Principle component analysis

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.9, svd_solver='full')
>>> pca.fit(X)
```

```
>>> X = pca.transform(X)
```
create a principle component analysis model with preserving 90% explained variance.

Other optional parameters:
`n_components=K`: preserve at the most K features.

3.2.9 Latent Dirichlet allocation

```
>>> from sklearn.decomposition import LatentDirichletAllocation
>>> X = LatentDirichletAllocation(n_component=20, n_topics=10, max_iter=5, learning_method='
learning_offset=50.,random_state=0).fit_transform(X)
```
create a latent Dirichlet allocation model with 20 components, 10 topics, 5 iterations, using online learning and a learning offset of 50, and the starting value is generated with random seed 0.

4 sklearn-crfsuite

4.1 Introduction to sklearn-crfsuite

Sklearn-crfsuite is a wrapper of pycrfsuite which provides fully compatibility with scikit-learn: you can use other routines scikit-learn routines, such as model selection or feature reduction, then pass the result to sklearn-crfsuite for training.

The more detailed information of sklearn-crfsuite can be found **here**.

4.2 Frequently Used Models

The complete document of sklearn-crfsuite API can be found **here**.

4.2.1 Conditional random field

Here is an example of building a conditional random field model using CoNLL 2002 corpus. To run this program, you have to install NLTK for Python.

```
from itertools import chain
import nltk
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import LabelBinarizer
import sklearn
import pycrfsuite
train_sents = list(nltk.corpus.conll2002.iob_sents('esp.train'))
test_sents = list(nltk.corpus.conll2002.iob_sents('esp.testb'))
```
import the training and testing data;
```
def word2features(sent, i):
  word = sent[i][0]
  postag = sent[i][1]
  features = ['bias',
    'word.lower=' + word.lower(),
    'word[-3:]=' + word[-3:],
```

```
    'word[-2:]=' + word[-2:],
    'word.isupper=%s' % word.isupper(),
    'word.istitle=%s' % word.istitle(),
    'word.isdigit=%s' % word.isdigit(),
    'postag=' + postag,
    'postag[:2]=' + postag[:2],]
  if i > 0:
    word1 = sent[i-1][0]
    postag1 = sent[i-1][1]
    features.extend(['-1:word.lower=' + word1.lower(),
      '-1:word.istitle=%s' % word1.istitle(),
      '-1:word.isupper=%s' % word1.isupper(),
      '-1:postag=' + postag1,
      '-1:postag[:2]=' + postag1[:2],])
  else:
    features.append('BOS')
  if i < len(sent)-1:
    word1 = sent[i+1][0]
    postag1 = sent[i+1][1]
    features.extend(['+1:word.lower=' + word1.lower(),
      '+1:word.istitle=%s' % word1.istitle(),
      '+1:word.isupper=%s' % word1.isupper(),
      '+1:postag=' + postag1,
      '+1:postag[:2]=' + postag1[:2],])
  else:
    features.append('EOS')
  return features
```

define the features: word identity, word suffix, word shape and word POS tag; also, some information from nearby words is used;

```
def sent2features(sent):
  return [word2features(sent, i) for i in range(len(sent))]

def sent2labels(sent):
  return [label for token, postag, label in sent]

def sent2tokens(sent):
  return [token for token, postag, label in sent]
```

define functions that get features, labels and tokens;

```
X_train = [sent2features(s) for s in train_sents]
y_train = [sent2labels(s) for s in train_sents]
X_test = [sent2features(s) for s in test_sents]
y_test = [sent2labels(s) for s in test_sents]
trainer = pycrfsuite.Trainer(verbose=False)
for xseq, yseq in zip(X_train, y_train):
  trainer.append(xseq, yseq)
trainer.set_params('c1':  1.0, 'c2':  1e-3, 'max_iterations':  50, 'feature.possible_transit
True)
trainer.train('conll2002-esp.crfsuite')
```

train the model using pycrfsuite.Trainer, load the training data and call 'train' method. First, create pycrfsuite.Trainer and load the training data to it. Second, set the training parameters as L-BFGS training algorithm with elastic net regularization. Finally, save the model to a file;

```
tagger = pycrfsuite.Tagger()
tagger.open('conll2002-esp.crfsuite')
example_sent = test_sents[0]
print(' '.join(sent2tokens(example_sent)), end='\n\n')
```

```
print('Predicted:', ' '.join(tagger.tag(sent2features(example_sent))))
print('Correct:  ', ' '.join(sent2labels(example_sent)))
```
open the model and tag an instance.

5 Pandas

5.1 Introduction to Pandas

Pandas is a software library of Python for data manipulation and analysis. In particular, Pandas offers data structures and operations for manipulating numerical tables and statistical time series. Pandas requires NumPy and SciPy.

5.2 Frequently Used Models

The complete document of Pandas API can be found **here**.

5.2.1 Statistical time series

Here is an example of building a statistical time series model with Pandas. To run this program, you have to install xarray and seaborn for Python.

```
import numpy as np
import pandas as pd
import seaborn as sns
import xarray as xr

np.random.seed(10)
times = pd.date_range('2000-01-01', '2001-12-31', name='time')
annual_cycle = np.sin(2 * np.pi * (times.dayofyear.values / 365.25 - 0.28))
base = 10 + 15 * annual_cycle.reshape(-1, 1)
tmin_values = base + 3 * np.random.randn(annual_cycle.size, 3)
tmax_values = base + 10 + 3 * np.random.randn(annual_cycle.size, 3)
ds = xr.Dataset({'tmin':  (('time', 'location'), tmin_values),
  'tmax':  (('time', 'location'), tmax_values)},
  {'time':  times, 'location':  ['IA', 'IN', 'IL']})
df = ds.to_dataframe()
ds.mean(dim='location').to_dataframe().plot()
sns.pairplot(df.reset_index(), vars=ds.data_vars)
```
examine a weather dataset from 2000 to 2001 by visualizing its minimums and maximums;
```
freeze = (ds['tmin'] <= 0).groupby('time.month').mean('time')
freeze.to_pandas().plot()
```
calculate the probability of freeze by calendar month;
```
monthly_avg = ds.resample(time='1MS').mean()
monthly_avg.sel(location='IA').to_dataframe().plot(style='s-')
```
calculate the monthly average;
```
climatology = ds.groupby('time.month').mean('time')
anomalies = ds.groupby('time.month') - climatology
anomalies.mean('location').to_dataframe()[['tmin', 'tmax']].plot()
```
calculate the monthly anomalies.

6 TensorFlow

6.1 Introduction to TensorFlow

TensorFlow is an open-source software library for data flow programming for machine learning applications, especially advanced neural networks (convolutional, recurrent, deep reinforcement learning, etc). It is used for both research and production at Google. In Windows platform, TensorFlow is only compatible with Python 3.

6.2 Frequently Used Models

The complete document of TensorFlow can be found **here**.

6.2.1 Conditional random field

6.2.2 Convolutional neural network

Here is an example of building a convolutional neural network with TensorFlow.

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data/', one_hot=True)
```
load internal MNIST data, a dataset of classifying 10 digits from 28x28 gray-scale images;
```
xs = tf.placeholder(tf.float32, [None, 784])
ys = tf.placeholder(tf.float32, [None, 10])
x_image = tf.reshape(xs, [-1, 28, 28, 1])
```
set the format of features (with 784 columns and unknown numbers of rows) and labels (with 1 column and unknown numbers of rows). And reshape the feature matrix into unknown numbers of 28x28x1 images ("x1" means gray-scale, and "x3" means color);
```
W_conv1 = tf.Variable(tf.truncated_normal([5, 5, 1, 32], 0, 0.1))
b_conv1 = tf.Variable(tf.zeros(32))
```
set the convolutional layer: 32 features where each is a 5x5 convolutional feature using same padding;
```
h_conv1 = tf.nn.relu(tf.nn.conv2d(x_image, W_conv1, [1, 1, 1, 1], 'SAME') + b_conv1)
```
set the activator: rectified linear unit; `h_pool1 = tf.nn.max_pool(h_conv1, [1, 2, 2, 1], [1, 2, 2, 1], 'SAME')`
set the pooling layer: pick the max value from each adjacent 2x2 convolutions using same padding;
```
h_pool1_flat = tf.reshape(h_pool1, [-1, 14*14*32])
```
flatten the convolutions to make them connect with non-convolutional layers;
```
W_fc1 = tf.Variable(tf.truncated_normal([14*14*32, 1024], 0, 0.1))
b_fc1 = tf.Variable(tf.zeros(1024))
h_fc1 = tf.nn.relu(tf.matmul(h_pool1_flat, W_fc1) + b_fc1)
```
set the non-convolutional layer: 1024 features;
```
W_fc2 = tf.Variable(tf.truncated_normal([1024, 10], 0, 0.1))
b_fc2 = tf.Variable(tf.zeros(10))
prediction = tf.nn.softmax(tf.matmul(h_fc1, W_fc2) + b_fc2)
```
set the output layer (clearly also non-convolutional);
```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(ys*tf.log(prediction), [1]))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
```
set the loss function as cross entropy (a popular criterion for multi-class classification task) and minimization method as Adam optimization (much faster than gradient descent);
```
sess = tf.Session()
```

```
sess.run(tf.global_variables_initializer())
```
create a session and initialize. In TensorFlow, the moment you create a session is the moment that the task starts running. All the statements prior to the session are just the framework.

```
for i in range(501):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict=xs:  batch_xs, ys:  batch_ys)
  if i % 50 == 0:
    y_pre = sess.run(prediction, feed_dict=xs:  mnist.test.images)
    res = tf.equal(tf.argmax(y_pre, 1), tf.argmax(mnist.test.labels, 1))
    accuracy = tf.reduce_mean(tf.cast(res, tf.float32))
    print(sess.run(accuracy, feed_dict=xs:  mnist.test.images))
```
Train for 500 steps, with a batch size of 100 for each step.

6.2.3 Recurrent neural network

Here is an example of building recurrent neural network to predict trigonometric function.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
BATCH_START = 0
TIME_STEPS = 20
BATCH_SIZE = 50
INPUT_SIZE = 1
OUTPUT_SIZE = 1
CELL_SIZE = 10
LR = 0.006
```
set the properties of the recurrent neural network: starting data index, iterations, batch size, input and output dimensions, hidden unit numbers and learning rate;

```
def get_batch():
  global BATCH_START, TIME_STEPS
  xs = np.arange(BATCH_START, BATCH_START+TIME_STEPS*BATCH_SIZE).reshape((BATCH_SIZE,
TIME_STEPS)) / (10*np.pi)
  seq = np.sin(xs)
  res = np.cos(xs)
  BATCH_START += TIME_STEPS
  return [seq[:, :, np.newaxis], res[:, :, np.newaxis], xs]
```
define a routine that fetch data;

```
class LSTMRNN(object):
  def __init__(self, n_steps, input_size, output_size, cell_size, batch_size):
    self.n_steps = n_steps
    self.input_size = input_size
    self.output_size = output_size
    self.cell_size = cell_size
    self.batch_size = batch_size
    with tf.name_scope('inputs'):
      self.xs = tf.placeholder(tf.float32, [None, n_steps, input_size], name='xs')
      self.ys = tf.placeholder(tf.float32, [None, n_steps, output_size], name='ys')
    with tf.variable_scope('in_hidden'):
      self.add_input_layer()
    with tf.variable_scope('LSTM_cell'):
      self.add_cell()
    with tf.variable_scope('out_hidden'):
```

```
      self.add_output_layer()
    with tf.name_scope('cost'):
      self.compute_cost()
    with tf.name_scope('train'):
      self.train_op = tf.train.AdamOptimizer(LR).minimize(self.cost)
```
define the framework of the recurrent neural network;
```
  def add_input_layer(self,):
    l_in_x = tf.reshape(self.xs, [-1, self.input_size], name='2_2D')
    Ws_in = self._weight_variable([self.input_size, self.cell_size])
    bs_in = self._bias_variable([self.cell_size,])
    with tf.name_scope('Wx_plus_b'):
      l_in_y = tf.matmul(l_in_x, Ws_in) + bs_in
    self.l_in_y = tf.reshape(l_in_y, [-1, self.n_steps, self.cell_size], name='2_3D')
```
define the routine to add the input layer;
```
  def add_cell(self):
    lstm_cell = tf.contrib.rnn.BasicLSTMCell(self.cell_size, forget_bias=1.0, state_is_tuple='
    with tf.name_scope('initial_state'):
      self.cell_init_state = lstm_cell.zero_state(self.batch_size, dtype=tf.float32)
    self.cell_outputs, self.cell_final_state = tf.nn.dynamic_rnn(
      lstm_cell, self.l_in_y, initial_state=self.cell_init_state, time_major=False)
```
define the routine to add one hidden unit;
```
  def add_output_layer(self):
    l_out_x = tf.reshape(self.cell_outputs, [-1, self.cell_size], name='2_2D')
    Ws_out = self._weight_variable([self.cell_size, self.output_size])
    bs_out = self._bias_variable([self.output_size, ])
    with tf.name_scope('Wx_plus_b'):
      self.pred = tf.matmul(l_out_x, Ws_out) + bs_out
```
define the routine to add the output layer;
```
  def compute_cost(self):
    losses = tf.contrib.legacy_seq2seq.sequence_loss_by_example(
      [tf.reshape(self.pred, [-1], name='reshape_pred')],
      [tf.reshape(self.ys, [-1], name='reshape_target')],
      [tf.ones([self.batch_size * self.n_steps], dtype=tf.float32)],
      average_across_timesteps=True,
      softmax_loss_function=self.ms_error,
      name='losses')
    with tf.name_scope('average_cost'):
      self.cost = tf.div(
        tf.reduce_sum(losses, name='losses_sum'),
        self.batch_size,
        name='average_cost')
      tf.summary.scalar('cost', self.cost)

  def ms_error(self, y_target, y_pre):
    return tf.square(tf.sub(y_target, y_pre))

  def _weight_variable(self, shape, name='weights'):
    initializer = tf.random_normal_initializer(mean=0., stddev=1.,)
    return tf.get_variable(shape=shape, initializer=initializer, name=name)

  def _bias_variable(self, shape, name='biases'):
    initializer = tf.constant_initializer(0.1)
    return tf.get_variable(name=name, shape=shape, initializer=initializer)
```
define other routines: loss function, weight and bias;
```
if __name__ == '__main__':
```

```
model = LSTMRNN(TIME_STEPS, INPUT_SIZE, OUTPUT_SIZE, CELL_SIZE, BATCH_SIZE)
sess = tf.Session()
sess.run(tf.global_variables_initializer())
for i in range(200):
  seq, res, xs = get_batch()
  if i == 0:
    feed_dict = {
        model.xs:  seq,
        model.ys:  res,}
  else:
    feed_dict = {
      model.xs:  seq,
      model.ys:  res,
      model.cell_init_state:  state}
  _, cost, state, pred = sess.run(
    [model.train_op, model.cost, model.cell_final_state, model.pred],
    feed_dict=feed_dict)
  if i % 20 == 0:
    print('cost:  ', round(cost, 4))
```

start training for 200 steps and print the result for every 20 steps.

6.2.4 Deep Q learning

Here is an example of building a deep Q learning (deep reinforcement learning) game framework with TensorFlow. To run this program, you have to install PyGame and OpenCV for Python.

```
import pygame
import random
import numpy as np
from collections import deque
import tensorflow as tf
import cv2
import time
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
SCREEN_SIZE = [320, 400]
BAR_SIZE = [50, 5]
BALL_SIZE = [15, 15]
```

set the attribute of the game: the size of the screen, the bar and the ball;

```
MOVE_STAY = [1, 0, 0]
MOVE_LEFT = [0, 1, 0]
MOVE_RIGHT = [0, 0, 1]
```

set the three choices of the bar moving;

```
class Game(object):
  def __init__(self):
    pygame.init()
    self.clock = pygame.time.Clock()
    self.screen = pygame.display.set_mode(SCREEN_SIZE)
    pygame.display.set_caption('Simple Game')
    self.ball_pos_x = SCREEN_SIZE[0]//2 - BALL_SIZE[0]/2
    self.ball_pos_y = SCREEN_SIZE[1]//2 - BALL_SIZE[1]/2
    self.ball_dir_x = -1
    self.ball_dir_y = -1
```

```
    self.ball_pos = pygame.Rect(self.ball_pos_x, self.ball_pos_y, BALL_SIZE[0], BALL_SIZE[1])
    self.bar_pos_x = SCREEN_SIZE[0]//2 - BAR_SIZE[0]//2
    self.bar_pos=pygame.Rect(self.bar_pos_x,SCREEN_SIZE[1]-BAR_SIZE[1],BAR_SIZE[0],BAR_SIZE[1]
```

set the initial state of the game (ball position, ball direction, bar position, etc.);

```
def step(self, action):
  if action == MOVE_LEFT:
    self.bar_pos_x = self.bar_pos_x - 2
  elif action == MOVE_RIGHT:
    self.bar_pos_x = self.bar_pos_x + 2
  else:
    pass
  if self.bar_pos_x < 0:
    self.bar_pos_x = 0
  if self.bar_pos_x > SCREEN_SIZE[0] - BAR_SIZE[0]:
    self.bar_pos_x = SCREEN_SIZE[0] - BAR_SIZE[0]
```

calculate the new position of the bar;

```
  self.screen.fill(BLACK)
  self.bar_pos.left = self.bar_pos_x
  pygame.draw.rect(self.screen, WHITE, self.bar_pos)
  self.ball_pos.left += self.ball_dir_x * 2
  self.ball_pos.bottom += self.ball_dir_y * 3
  pygame.draw.rect(self.screen, WHITE, self.ball_pos)
```

update the position of the bar and the ball on the screen;

```
  if self.ball_pos.top <= 0 or self.ball_pos.bottom >= (SCREEN_SIZE[1] - BAR_SIZE[1]+1):
    self.ball_dir_y = self.ball_dir_y * -1
  if self.ball_pos.left <= 0 or self.ball_pos.right >= (SCREEN_SIZE[0]):
    self.ball_dir_x = self.ball_dir_x * -1
```

change the direction of the ball when hitting the edge of the screen;

```
  reward = 0
  if self.bar_pos.top <= self.ball_pos.bottom and (self.bar_pos.left < self.ball_pos.right
and self.bar_pos.right > self.ball_pos.left):
    reward = 1
  elif self.bar_pos.top <= self.ball_pos.bottom and (self.bar_pos.left > self.ball_pos.right
or self.bar_pos.right < self.ball_pos.left):
    reward = -1
```

calculate the reward;

```
  screen_image = pygame.surfarray.array3d(pygame.display.get_surface())
  pygame.display.update()
  return reward, screen_image
```

reprint the screen;

```
LEARNING_RATE = 0.99
INITIAL_EPSILON = 1.0
FINAL_EPSILON = 0.05
EXPLORE = 500000
OBSERVE = 50000
REPLAY_MEMORY = 500000
BATCH = 100
output = 3
input_image = tf.placeholder("float", [None, 80, 100, 4])
action = tf.placeholder("float", [None, output])
```

set the properties of the training: learning rate, precision, observance numbers, history numbers, batch size, class numbers and the format of the inputs;

```
def convolutional_neural_network(input_image):
  weights = {'w_conv1':tf.Variable(tf.zeros([8, 8, 4, 32])),
```

```
      'w_conv2':tf.Variable(tf.zeros([4, 4, 32, 64])),
      'w_conv3':tf.Variable(tf.zeros([3, 3, 64, 64])),
      'w_fc4':tf.Variable(tf.zeros([3456, 784])),
      'w_out':tf.Variable(tf.zeros([784, output]))}
  biases = {'b_conv1':tf.Variable(tf.zeros([32])),
      'b_conv2':tf.Variable(tf.zeros([64])),
      'b_conv3':tf.Variable(tf.zeros([64])),
      'b_fc4':tf.Variable(tf.zeros([784])),
      'b_out':tf.Variable(tf.zeros([output]))}
  conv1 = tf.nn.relu(tf.nn.conv2d(input_image, weights['w_conv1'], strides = [1,
4, 4, 1], padding = "VALID") + biases['b_conv1'])
  conv2 = tf.nn.relu(tf.nn.conv2d(conv1, weights['w_conv2'], strides = [1, 2, 2,
1], padding = "VALID") + biases['b_conv2'])
  conv3 = tf.nn.relu(tf.nn.conv2d(conv2, weights['w_conv3'], strides = [1, 1, 1,
1], padding = "VALID") + biases['b_conv3'])
  conv3_flat = tf.reshape(conv3, [-1, 3456])
  fc4 = tf.nn.relu(tf.matmul(conv3_flat, weights['w_fc4']) + biases['b_fc4'])
  output_layer = tf.matmul(fc4, weights['w_out']) + biases['b_out']
  return output_layer
```

define the structure of the neural network: 3 convolutional layers, 1 fully-connected layer and the output layer;

```
def train_neural_network(input_image):
  predict_action = convolutional_neural_network(input_image)
  argmax = tf.placeholder("float", [None, output])
  gt = tf.placeholder("float", [None])
  action = tf.reduce_sum(tf.multiply(predict_action, argmax), reduction_indices =
1)
  cost = tf.reduce_mean(tf.square(action - gt))
  optimizer = tf.train.AdamOptimizer(1e-6).minimize(cost)
  game = Game()
  D = deque()
```

set the loss function and optimization method;

```
  _, image = game.step(MOVE_STAY)
  image = cv2.cvtColor(cv2.resize(image, (100, 80)), cv2.COLOR_BGR2GRAY)
  ret, image = cv2.threshold(image, 1, 255, cv2.THRESH_BINARY)
  input_image_data = np.stack((image, image, image, image), axis = 2)
convert the image to binary attributes;
  with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())
    saver = tf.train.Saver()
    n = 0
    while True:
      time.sleep(0.001)
      action_t = predict_action.eval(feed_dict = input_image :  [input_image_data])[0]
      argmax_t = np.zeros([output], dtype=np.int)
      if(random.random() <= INITIAL_EPSILON):
        maxIndex = random.randrange(output)
      else:
        maxIndex = np.argmax(action_t)
      argmax_t[maxIndex] = 1
      if epsilon > FINAL_EPSILON:
        epsilon -= (INITIAL_EPSILON - FINAL_EPSILON) / EXPLORE
```

start the optimization;

```
      #for event in pygame.event.get():
      #  if event.type == QUIT:
```

```
      #    pygame.quit()
      #    sys.exit()
```
this part is only needed for Mac OS;
```
      reward, image = game.step(list(argmax_t))
      image = cv2.cvtColor(cv2.resize(image, (100, 80)), cv2.COLOR_BGR2GRAY)
      ret, image = cv2.threshold(image, 1, 255, cv2.THRESH_BINARY)
      image = np.reshape(image, (80, 100, 1))
      input_image_data1 = np.append(image, input_image_data[:, :, 0:3], axis = 2)
      D.append((input_image_data, argmax_t, reward, input_image_data1))
```
start exploitation and add to history;
```
      if len(D) > REPLAY_MEMORY:
        D.popleft()
      if n > OBSERVE:
        minibatch = random.sample(D, BATCH)
        input_image_data_batch = [d[0] for d in minibatch]
        argmax_batch = [d[1] for d in minibatch]
        reward_batch = [d[2] for d in minibatch]
        input_image_data1_batch = [d[3] for d in minibatch]
```
when exploitation is complete, start exploration;
```
        gt_batch = []
        out_batch = predict_action.eval(feed_dict = input_image :  input_image_data1_batch)
        for i in range(0, len(minibatch)):
          gt_batch.append(reward_batch[i] + LEARNING_RATE * np.max(out_batch[i]))
        optimizer.run(feed_dict = gt :  gt_batch, argmax :  argmax_batch, input_image
:  input_image_data_batch)
      input_image_data = input_image_data1
      n = n+1
      if n % 1000000 == 0:
        saver.save(sess, 'game.cpk', global_step = n)
      print(n, "epsilon:", epsilon, " " ,"action:", maxIndex, " " ,"reward:", reward)
```
evaluate the model and print the results.