Theano: Theano is a popular python library that is used to define, evaluate, and optimize mathematical expressions involving multi-dimensional arrays in an efficient manner. It is achieved by optimizing the utilization of CPU and GPU.

```
# Python program using Theano
# for computing a Logistic
# Function

import theano
import theano.tensor as T
x = T.dmatrix('x')
s = 1 / (1 + T.exp(-x))
logistic = theano.function([x], s)
logistic([[0, 1], [-1, -2]])
```

Output:

```
array([[0.5, 0.73105858],

        [0.26894142, 0.11920292]])
```

TensorFlow: TensorFlow is a very popular open-source library for high performance numerical computation developed by the Google Brain team in Google. As the name suggests, Tensorflow is a framework that involves defining and running computations involving tensors.

```
#  Python program using TensorFlow
#  for multiplying two arrays

# import `tensorflow`
import tensorflow as tf

# Initialize two constants
x1 = tf.constant([1, 2, 3, 4])
x2 = tf.constant([5, 6, 7, 8])

# Multiply
result = tf.multiply(x1, x2)

# Initialize the Session
sess = tf.Session()
```

```
# Print the result
print(sess.run(result))

# Close the session
sess.close()
```

Output:

```
[ 5 12 21 32]
```

Keras: Keras is a very popular Machine Learning library for Python. It is a high-level neural networks API capable of running on top of TensorFlow, CNTK, or Theano. It can run seamlessly on both CPU and GPU. Keras makes it really for ML beginners to build and design a Neural Network.

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential()
model.add(Dense(64, activation='relu', input_dim=50)) #input shape of 50
model.add(Dense(28, activation='relu')) #input shape of 50
model.add(Dense(10, activation='softmax'))
```

PyTorch: PyTorch is a popular open-source Machine Learning library for Python based on Torch, which is an open-source Machine Learning library that is implemented in C with a wrapper in Lua. It has an extensive choice of tools and libraries that support Computer Vision, Natural Language Processing(NLP), and many more ML programs.

```
# Python program using PyTorch
# for defining tensors fit a
# two-layer network to random
# data and calculating the loss

import torch


dtype = torch.float
device = torch.device("cpu")
# device = torch.device("cuda:0") Uncomment this to run on GPU

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random input and output data
```

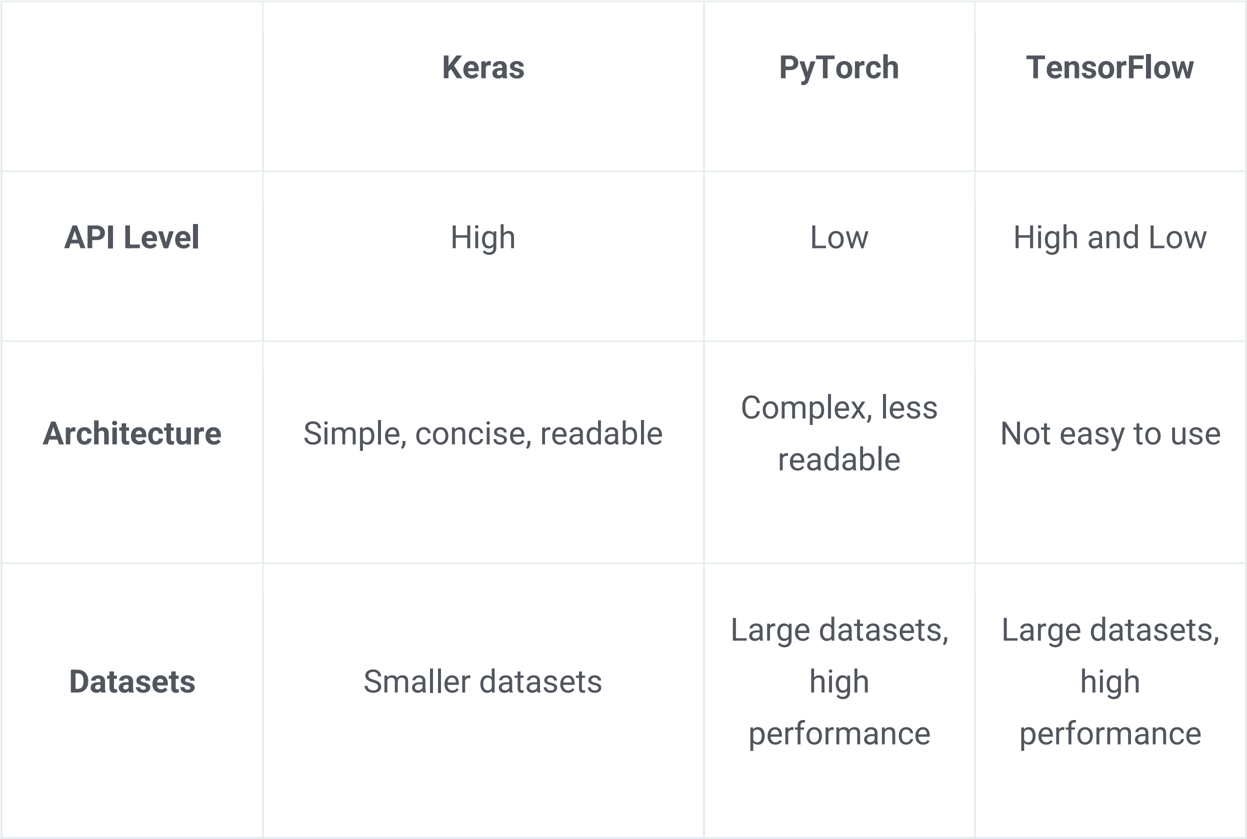
```python
x = torch.random(N, D_in, device=device, dtype=dtype)
y = torch.random(N, D_out, device=device, dtype=dtype)

# Randomly initialize weights
w1 = torch.random(D_in, H, device=device, dtype=dtype)
w2 = torch.random(H, D_out, device=device, dtype=dtype)

learning_rate = 1e-6
for t in range(500):
    # Forward pass: compute predicted y
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)

    # Compute and print loss
    loss = (y_pred - y).pow(2).sum().item()
    print(t, loss)

    # Backprop to compute gradients of w1 and w2 with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = x.t().mm(grad_h)

    # Update weights using gradient descent
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```

Output:

```
0 47168344.0

1 46385584.0

2 43153576.0

...

...

...

497 3.987660602433607e-05

498 3.945609932998195e-05

499 3.897604619851336e-05
```

Keras vs Tensorflow

Parameters	Keras	Tensorflow
Type	High-Level API Wrapper	Low-Level API
Complexity	Easy to use if you Python language	You need to learn the syntax of using some of Tensorflow function
Purpose	Rapid deployment for making model with standard layers	Allows you to make an arbitrary computational graph or model layers
Tools	Uses other API debug tool such as TFDBG	You can use Tensorboard visualization tools
Community	Large active communities	Large active communities and widely shared resources



	Keras	PyTorch	TensorFlow
API Level	High	Low	High and Low
Architecture	Simple, concise, readable	Complex, less readable	Not easy to use
Datasets	Smaller datasets	Large datasets, high performance	Large datasets, high performance

Debugging	Simple network, so debugging is not often needed	Good debugging capabilities	Difficult to conduct debugging
Does It Have Trained Models?	Yes	Yes	Yes
Popularity	Most popular	Third most popular	Second most popular
Speed	Slow, low performance	Fast, high-performance	Fast, high-performance
Written In	Python	Lua	C++, CUDA, Python

Conclusion:

Keras Is used in the scenarios where

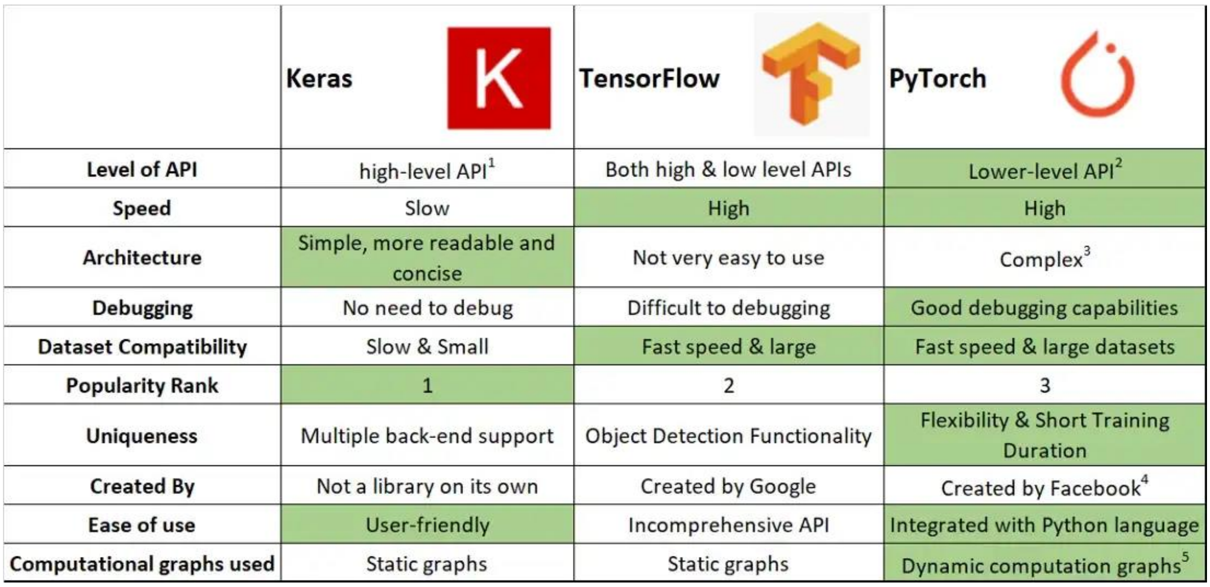
- Rapid Prototyping
- Small Dataset
- Multiple back-end support

Tensor Flow is used in the scenarios where

- Large Dataset
- High Performance
- Functionality
- Object Detection

PyTorch is used in scenarios where

- Flexibility
- Short Training Duration
- Debugging capabilities



	Keras	TensorFlow	PyTorch
Level of API	high-level API[1]	Both high & low level APIs	Lower-level API[2]
Speed	Slow	High	High
Architecture	Simple, more readable and concise	Not very easy to use	Complex[3]
Debugging	No need to debug	Difficult to debugging	Good debugging capabilities
Dataset Compatibility	Slow & Small	Fast speed & large	Fast speed & large datasets
Popularity Rank	1	2	3
Uniqueness	Multiple back-end support	Object Detection Functionality	Flexibility & Short Training Duration
Created By	Not a library on its own	Created by Google	Created by Facebook[4]
Ease of use	User-friendly	Incomprehensive API	Integrated with Python language
Computational graphs used	Static graphs	Static graphs	Dynamic computation graphs[5]