```
import tensorflow as tf
print(tf.__version__)
```

```
2.13.0
```

TensorFlow 2.0 + Keras 오버뷰

TensorFlow 2.0 + Keras 개발 배경

과거 TensorFlow 1.x + Keras 의 여러가지 문제점:

- TensorFlow를 사용한다는것은 정적인 계산 그래프를 조작함을 의미하는것으로, 일반적인 프로그래머들의 접근이 어려웠음.
- TensorFlow 1.x API 는 자유도는 높았지만 개발 생산성이 낮고 사용법이 어려웠음.
- Keras는 생산성이 높지만, 연구에 사용되기에는 자유도가 낮음.

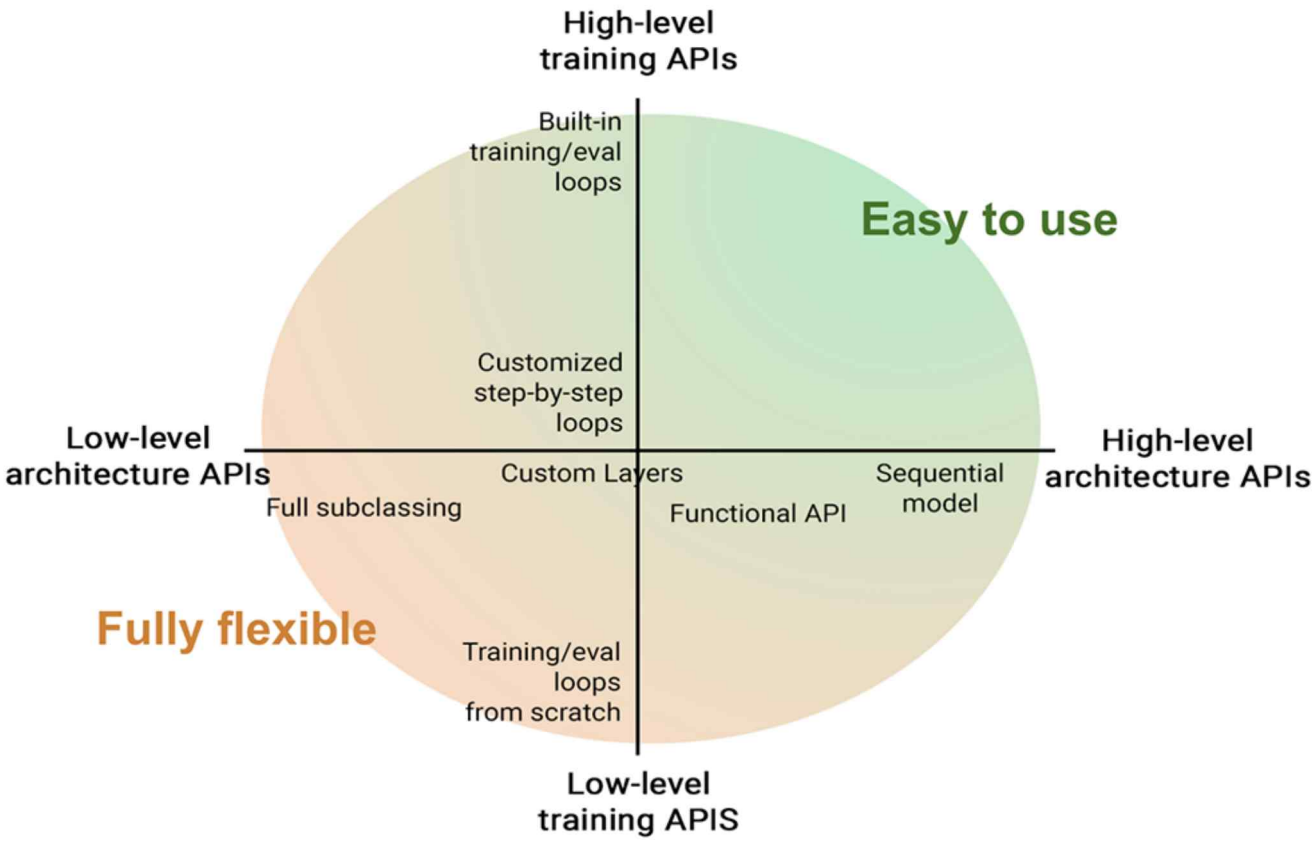
TensorFlow 2.0은 TensorFlow와 Keras를 새로이 디자인한 것으로, 위에서 언급된 문제점들을 대규모로 수정.

TensorFlow 2.0은 아래의 주요 아이디어를 가지고 개발됨:

- 사용자들이 계산을 eagerly하게 수행할 수 있게 해줍니다. 이는 Numpy의 사용법과 유사하게 하여 TensorFlow 2.0을 이용한 프로그래밍이 직관적고 파이토닉할 수 있게함.
- eagerly 하게 수행 하면서도 정적 그래프의 이점을 보존하여 TensorFlow를 빠르고, 분산 구조에서의 확장 가능하게 합니다.
- Keras를 딥러닝의 고수준 API로 채택하여, TensorFlow의 개발이 쉽고 높은 생산성을 가질 수 있게 함
- 고수준(쉬운 사용성, 부족한 유연성) API 에서부터 저수준(깊은 전문성, 뛰어난 유연성) API의 다양한 범위의 작업으로 Keras를 확장

Keras API

Tensorflow 2.0 + Keras는 고수준의 API와 저수준 API를의 유연하게 함께 사용 할 수있도록 합니다.



`Layer` 기본 클래스

Keras의 거의 모든것은 Layer 클래스로부터 파생됩니다.

- Layer는 상태(weights)와 `call` 메소드에 정의된 계산을 캡슐화 합니다.

```
from tensorflow.keras.layers import Layer

class Linear(Layer):
  """y = w.x + b"""

  def __init__(self, units=32):
      super(Linear, self).__init__()
      self.units = units

  def build(self, input_shape):
      self.w = self.add_weight(shape=(input_shape[-1], self.units),
                               initializer='random_normal',
                               trainable=True)
      self.b = self.add_weight(shape=(self.units,),
                               initializer='random_normal',
                               trainable=True)

  def call(self, inputs):
      return tf.matmul(inputs, self.w) + self.b
```

```
# 우리가 만든 Layer객체를 인스턴스화 합니다
linear_layer = Linear(4)
```

Layer 인스턴스는 마치 함수처럼 동작합니다. 몇 데이터에 대해서 이를 호출해 봅시다:

```
linear_layer(tf.ones((2, 2)))
```

```
<tf.Tensor: shape=(2, 4), dtype=float32, numpy=
array([[ 0.07553397, -0.07236704, -0.15786695,  0.01180486],
       [ 0.07553397, -0.07236704, -0.15786695,  0.01180486]],
      dtype=float32)>
```

`Layer` 클래스는 속성으로써 부여된 weights를 통해서, 가중치들을 추적합니다

```
# 가중치는 자동으로 `weights`라는 속성으로써 추적됩니다.
linear_layer.weights
```

```
[<tf.Variable 'linear/Variable:0' shape=(2, 4) dtype=float32, numpy=
 array([[ 0.08405342, -0.03657999,  0.01777976, -0.01185757],
        [-0.01075094,  0.00311644,  0.01849862,  0.08275844]],
       dtype=float32)>,
 <tf.Variable 'linear/Variable:0' shape=(4,) dtype=float32, numpy=array([-0.03117408,  0.02402791, -0.002667  , -0.04716046], dtype=float32)>]
```

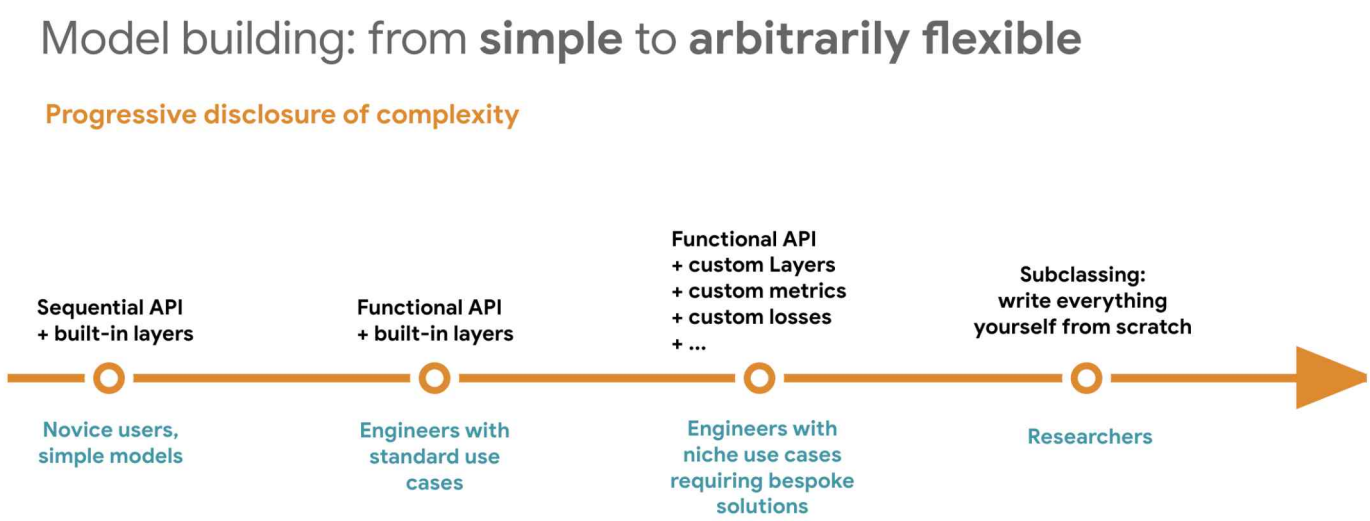
미리 정의된 Layer의 종류

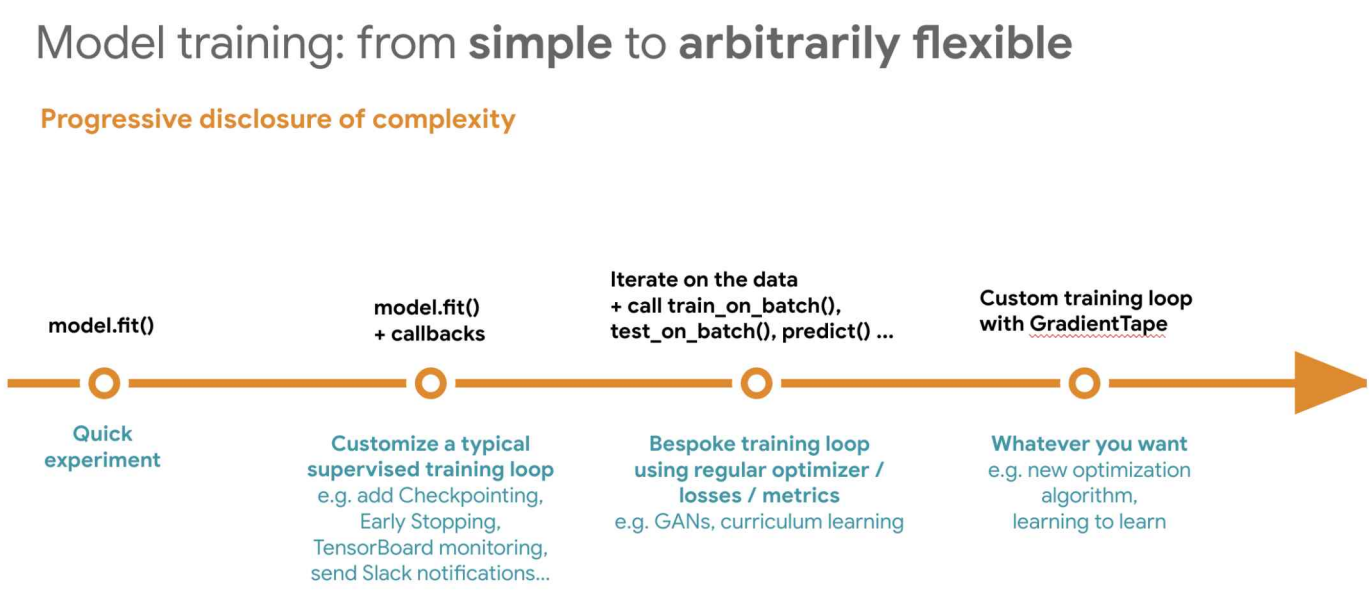
Keras는 넓은 범위의 미리 정의된 Layer의 종류를 제공하여 항상 여러분 스스로가 모든것을 구현하지 않아도 되도록끔 해 줍니다.

- Convolution layers
- Transposed convolutions
- Separateable convolutions
- Average and max pooling
- Global average and max pooling
- LSTM, GRU (with built-in cuDNN acceleration)
- BatchNormalization
- Dropout
- Attention
- ConvLSTM2D
- etc.

TensorFlow 2.0 + Keras 의 작업 흐름

- TensorFlow 2.0 + Keras의 중심 원칙은 "복잡도의 점진적인 공개"
- 매우 쉽게 시작할 수 고수준 API 부터, 많은 부분을 밑바닥에서 부터 구현해야 하는 저수준 API들이 존재
- 모델의 정의, 모델의 학습에도 중심 원칙을 위한 작업 흐름과 API가 존재


Model building: from simple to arbitrarily flexible
Progressive disclosure of complexity
Sequential API
+ built-in layers
Functional API
+ built-in layers
Functional API
+ custom Layers
+ custom metrics
+ custom losses
+ ...
Subclassing:
write everything
yourself from scratch
Novice users,
simple models
Engineers with
standard use
cases
Engineers with
niche use cases
requiring bespoke
solutions
Researchers


Model training: from simple to arbitrarily flexible
Progressive disclosure of complexity
model.fit()
model.fit()
+ callbacks
Iterate on the data
+ call train_on_batch(),
test_on_batch(), predict() ...
Custom training loop
with GradientTape
Quick
experiment
Customize a typical
supervised training loop
e.g. add Checkpointing,
Early Stopping,
TensorBoard monitoring,
send Slack notifications...
Bespoke training loop
using regular optimizer /
losses / metrics
e.g. GANs, curriculum learning
Whatever you want
e.g. new optimization
algorithm,
learning to learn