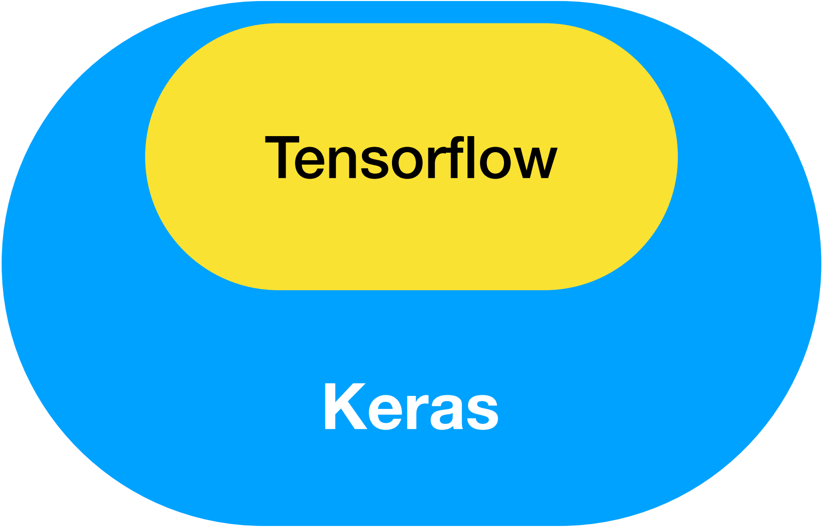
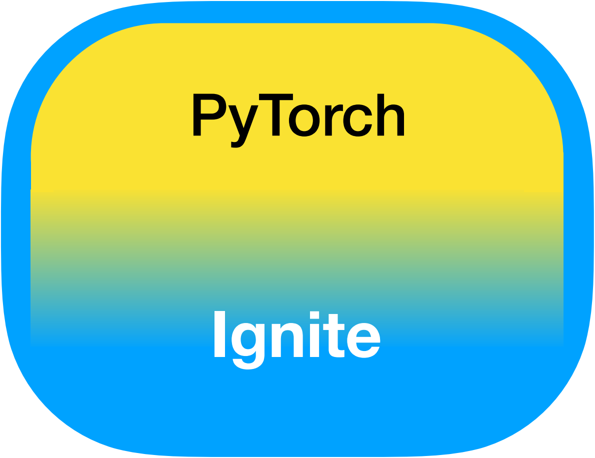
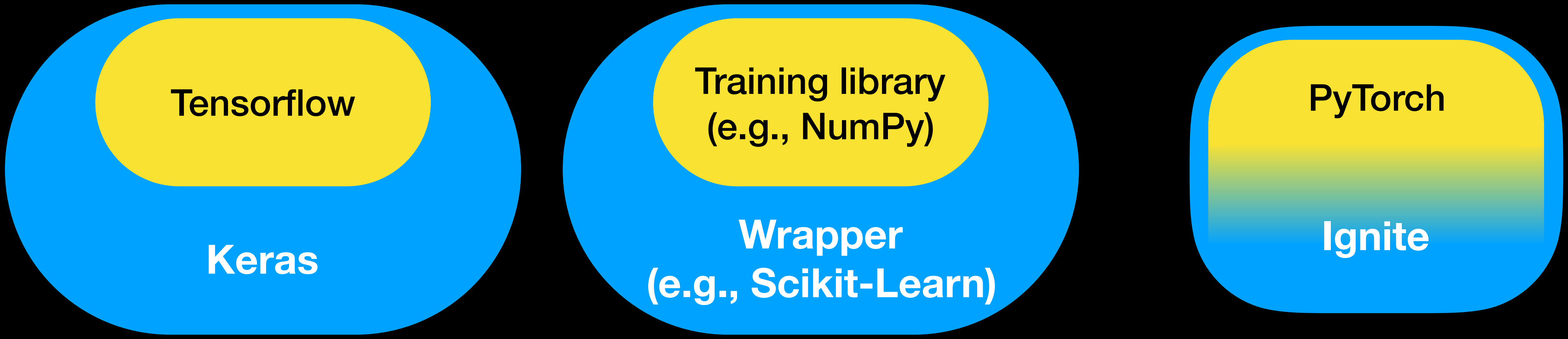
Training library (e.g., NumPy)


Tensorflow
Keras


PyTorch
Ignite

https://github.com/stsievert/talks

Every library has wrappers to create models easily.



Tensorflow

Keras

Training library (e.g., NumPy)

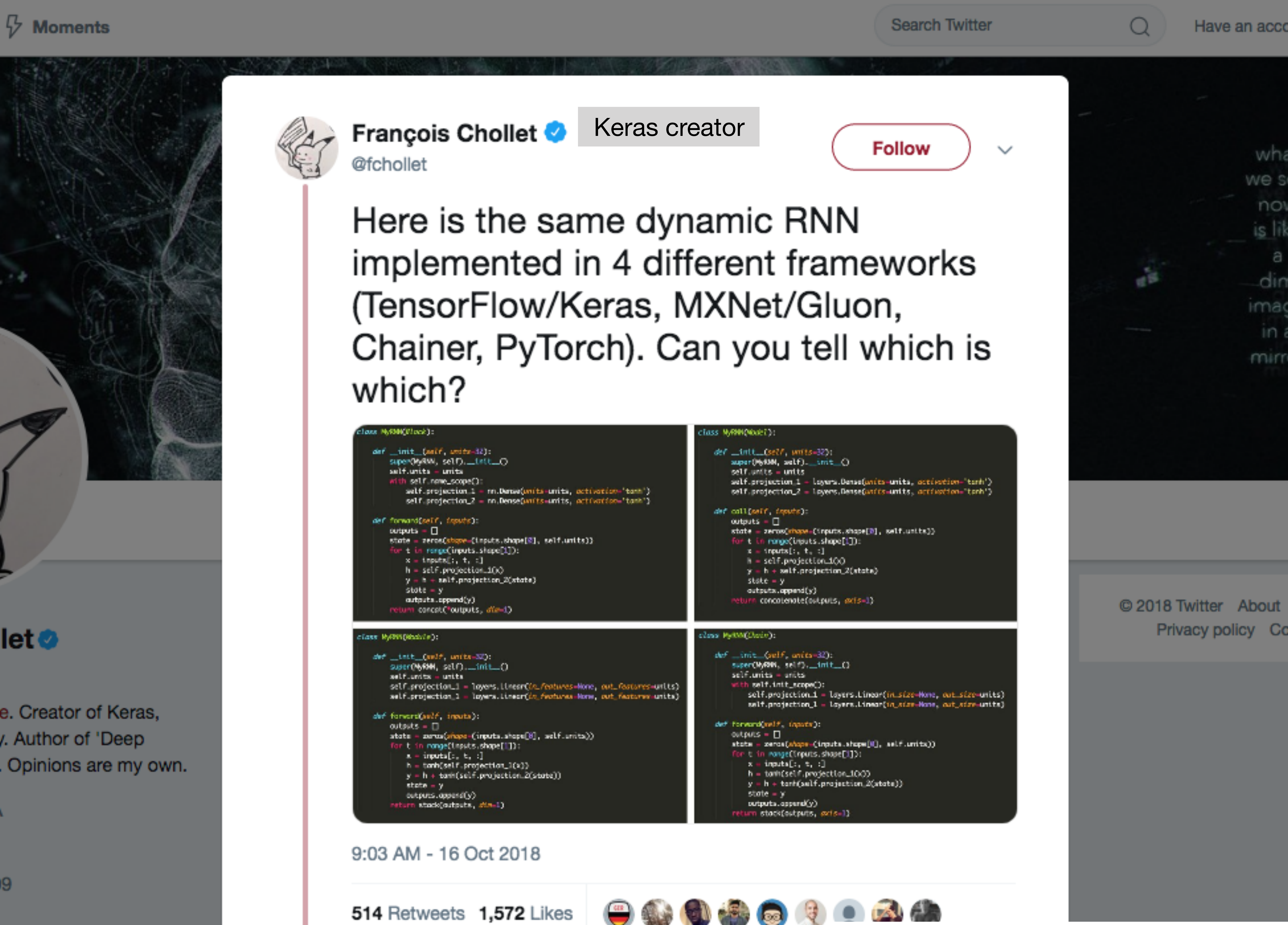
Wrapper (e.g., Scikit-Learn)

PyTorch

Ignite

PyTorch requires minimal wrapping

https://github.com/stsievert/talks



Keras creator

François Chollet
@fchollet

Here is the same dynamic RNN implemented in 4 different frameworks (TensorFlow/Keras, MXNet/Gluon, Chainer, PyTorch). Can you tell which is which?

```
class MyRNN(Block):

    def __init__(self, units=32):
        super(MyRNN, self).__init__()
        self.units = units
        with self.name_scope():
            self.projection_1 = nn.Dense(units=units, activation='tanh')
            self.projection_2 = nn.Dense(units=units, activation='tanh')

    def forward(self, inputs):
        outputs = []
        state = zeros(shape=(inputs.shape[0], self.units))
        for t in range(inputs.shape[1]):
            x = inputs[:, t, :]
            h = self.projection_1(x)
            y = h + self.projection_2(state)
            state = y
            outputs.append(y)
        return concat(*outputs, dim=1)
```

```
class MyRNN(Model):

    def __init__(self, units=32):
        super(MyRNN, self).__init__()
        self.units = units
        self.projection_1 = layers.Dense(units=units, activation='tanh')
        self.projection_2 = layers.Dense(units=units, activation='tanh')

    def call(self, inputs):
        outputs = []
        state = zeros(shape=(inputs.shape[0], self.units))
        for t in range(inputs.shape[1]):
            x = inputs[:, t, :]
            h = self.projection_1(x)
            y = h + self.projection_2(state)
            state = y
            outputs.append(y)
        return concatenate(outputs, axis=1)
```

```
class MyRNN(Module):

    def __init__(self, units=32):
        super(MyRNN, self).__init__()
        self.units = units
        self.projection_1 = layers.Linear(in_features=None, out_features=units)
        self.projection_1 = layers.Linear(in_features=None, out_features=units)

    def forward(self, inputs):
        outputs = []
        state = zeros(shape=(inputs.shape[0], self.units))
        for t in range(inputs.shape[1]):
            x = inputs[:, t, :]
            h = tanh(self.projection_1(x))
            y = h + tanh(self.projection_2(state))
            state = y
            outputs.append(y)
        return stack(outputs, dim=1)
```

```
class MyRNN(Chain):

    def __init__(self, units=32):
        super(MyRNN, self).__init__()
        self.units = units
        with self.init_scope():
            self.projection_1 = layers.Linear(in_size=None, out_size=units)
            self.projection_1 = layers.Linear(in_size=None, out_size=units)

    def forward(self, inputs):
        outputs = []
        state = zeros(shape=(inputs.shape[0], self.units))
        for t in range(inputs.shape[1]):
            x = inputs[:, t, :]
            h = tanh(self.projection_1(x))
            y = h + tanh(self.projection_2(state))
            state = y
            outputs.append(y)
        return stack(outputs, axis=1)
```

9:03 AM - 16 Oct 2018

514 Retweets 1,572 Likes

https://github.com/stsievert/talks