Tensorflow y Keras

Tensorflow vs Keras

Keras

```
# MODELO
model=keras.Sequential(
   keras.Dense(4,activation='
softmax')]

# ENTRENAMIENTO
model.compile(loss='mse',
optimizer='sgd')
model.fit(x,y,epochs=100)

# PREDICCION
y_predicted=model.predict(x)
```

Tensorflow

```
# MODELO

# Variables de entrada
varX = tf.placeholder("float")
varY = tf.placeholder("float")

# Parámetros
W = tf.Variable(np.random.randn())
b = tf.Variable(np.random.randn())

# Salida
y_abs = tf.add(tf.multiply(X, W), b)
y_pred=tf.softmax(y_abs)
```

Tensorflow vs Keras

Keras

```
# MODELO
model=keras.Sequential(
   keras.Dense(4,activation
='softmax')]

# ENTRENAMIENTO
model.compile(loss='mse',
optimizer='sgd')
model.fit(x,y,epochs=100)

# PREDICCION
y_predicted=model.predict(
x)
```

Tensorflow

```
# ENTRENAMIENTO
errors=tf.nn.sigmoid_cross_entropy_with_logi
ts(logits=y_pred, labels=varY)
error=tf.reduce_mean(errors)

opt = tf.train.GradientDescentOptimizer(
      learning_rate=0.001).minimize(error)

## varias líneas de sarasa ##

# epocas y batches "a mano"
epochs=1000
for i in range(epochs):
  for (batch_x, batch_y) in zip(x, y):
    sess.run(opt,
       feed_dict={varX:batch_x,varY:batch_y})
```

Tensorflow vs Keras

Keras

```
# MODELO
model=keras.Sequential(
   keras.Dense(4,activation
='softmax')]

# ENTRENAMIENTO
model.compile(loss='mse',
optimizer='sgd')
model.fit(x,y,epochs=100)

# PREDICCION
y_predicted=model.predict(
x)
```

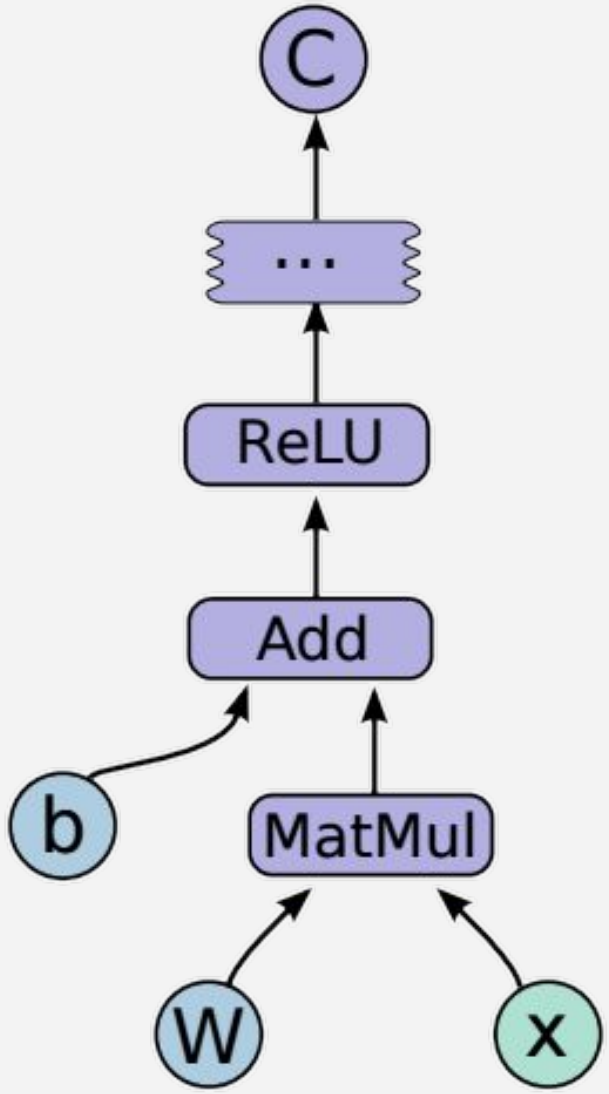
Tensorflow

```
# PREDICCION

y_pred_value = sess.run(y_pred,
feed_dict={varX:x})
```

Tensorflow - Grafos Declarativos

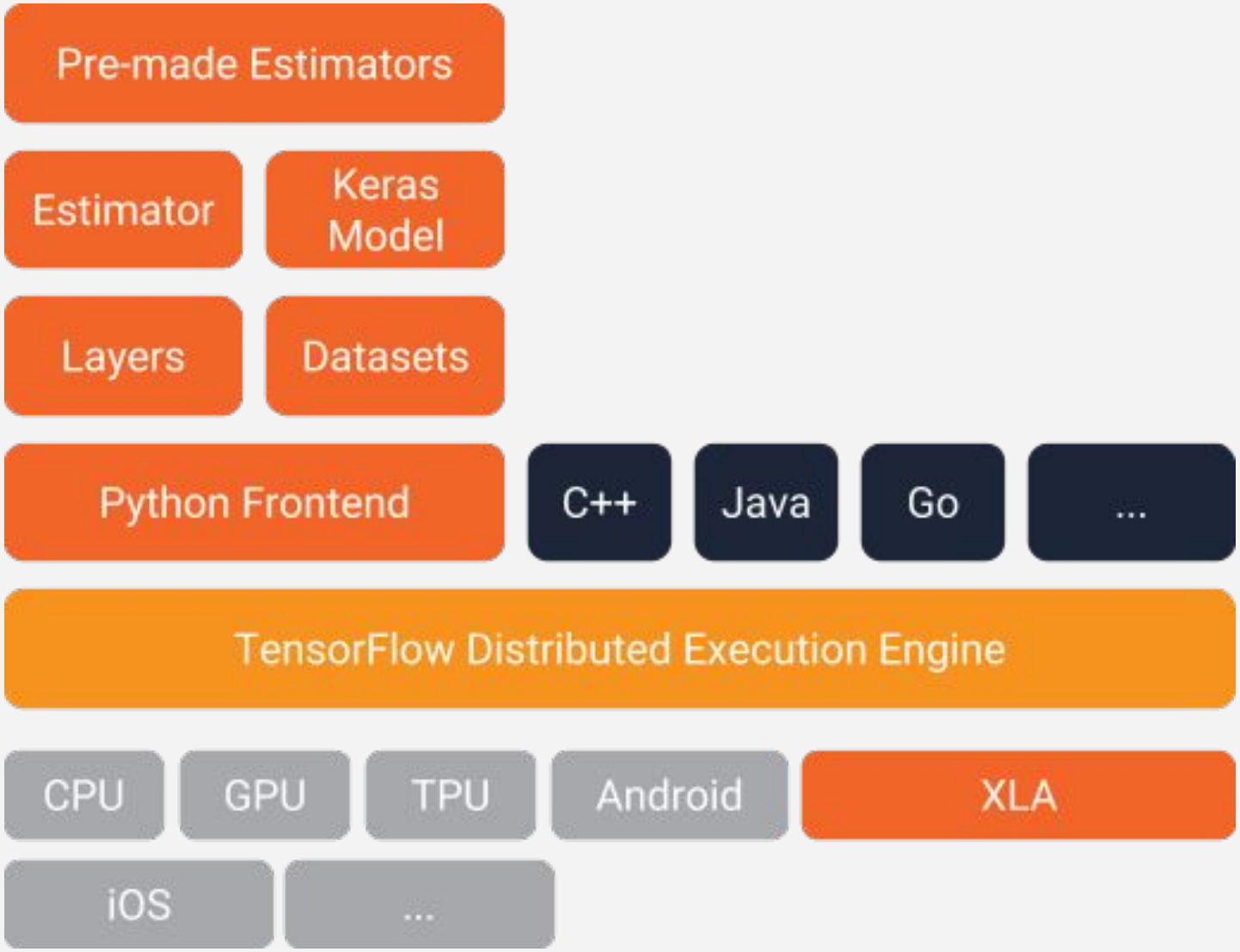
- Operaciones de tensorflow **declarativas**
  - NO se ejecutan inmediatamente
  - Definen grafos
  - Ejecución mediante sess.run(op)
- Desacoplan definición de la ejecución
  - Como una función!
  - Permite optimizar el grafo
- Ejemplo:

```
x = tf.placeholder("float")
W = tf.Variable(np.random.randn())
b = tf.Variable(np.random.randn())

# Salida
C = tf.nn.relu(tf.add(tf.multiply(x, W), b))
C_value = sess.run(C,feed_dict={x: INPUT })
# vectores numpy
```



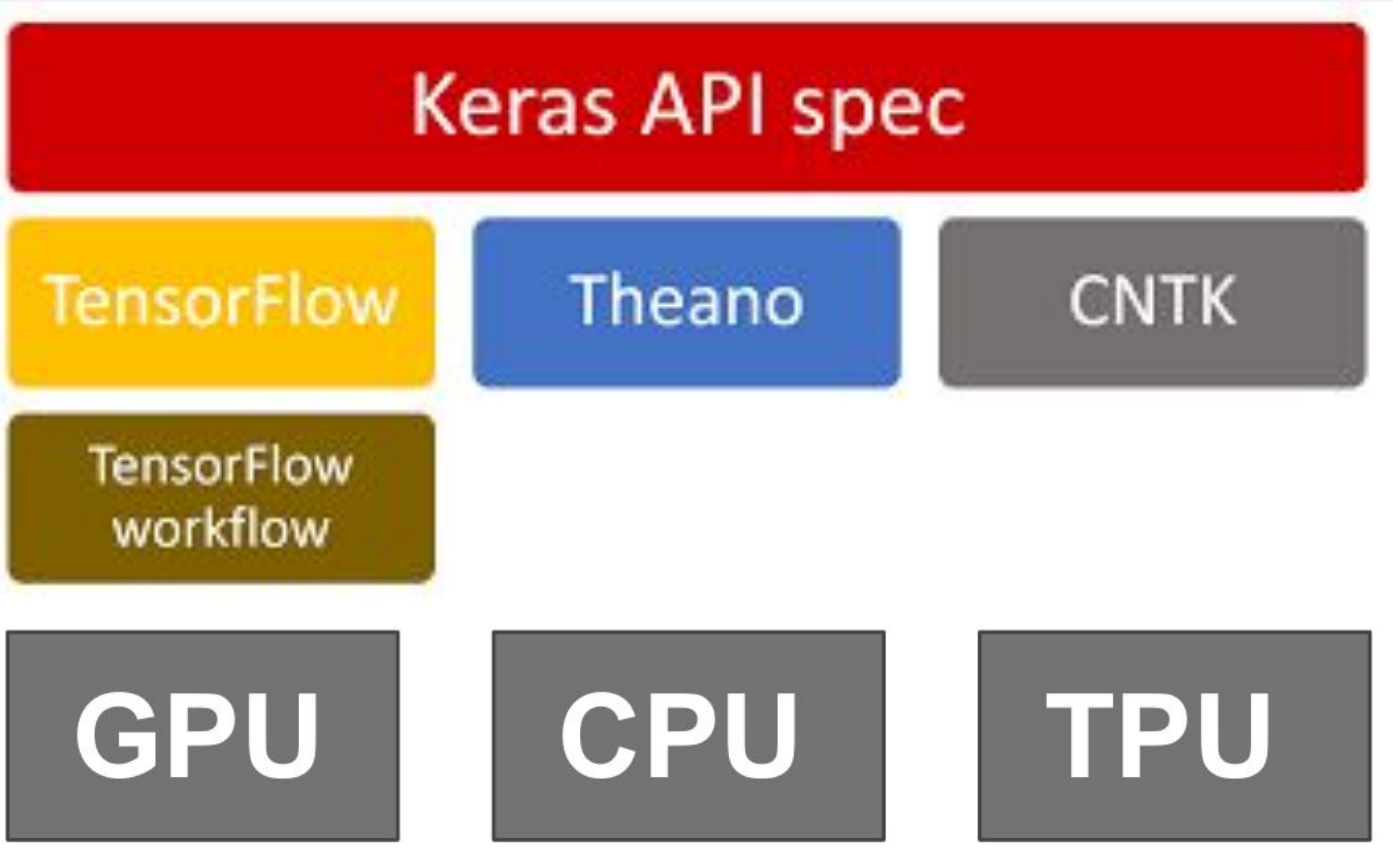
Tensorflow - Arquitectura

- Core en C++
- API en varios lenguajes
- Elementos
  - API de operadores
    - Grafo de computación
    - **Derivadas automáticas**
  - Implementación optimizada para cada dispositivo
  - Comunicación entre dispositivos



Keras - Arquitectura

- Keras: interfaz de alto nivel
  - Otras libs hacen el trabajo pesado (*backends*)
- Para definir Redes Neuronales
  - Y otros modelos diferenciables
- Provee API unificada para:
  - Cargar datos
  - Entrenar (fit)
  - Predecir (predict)
  - Evaluar (evaluate)




```
import keras
```

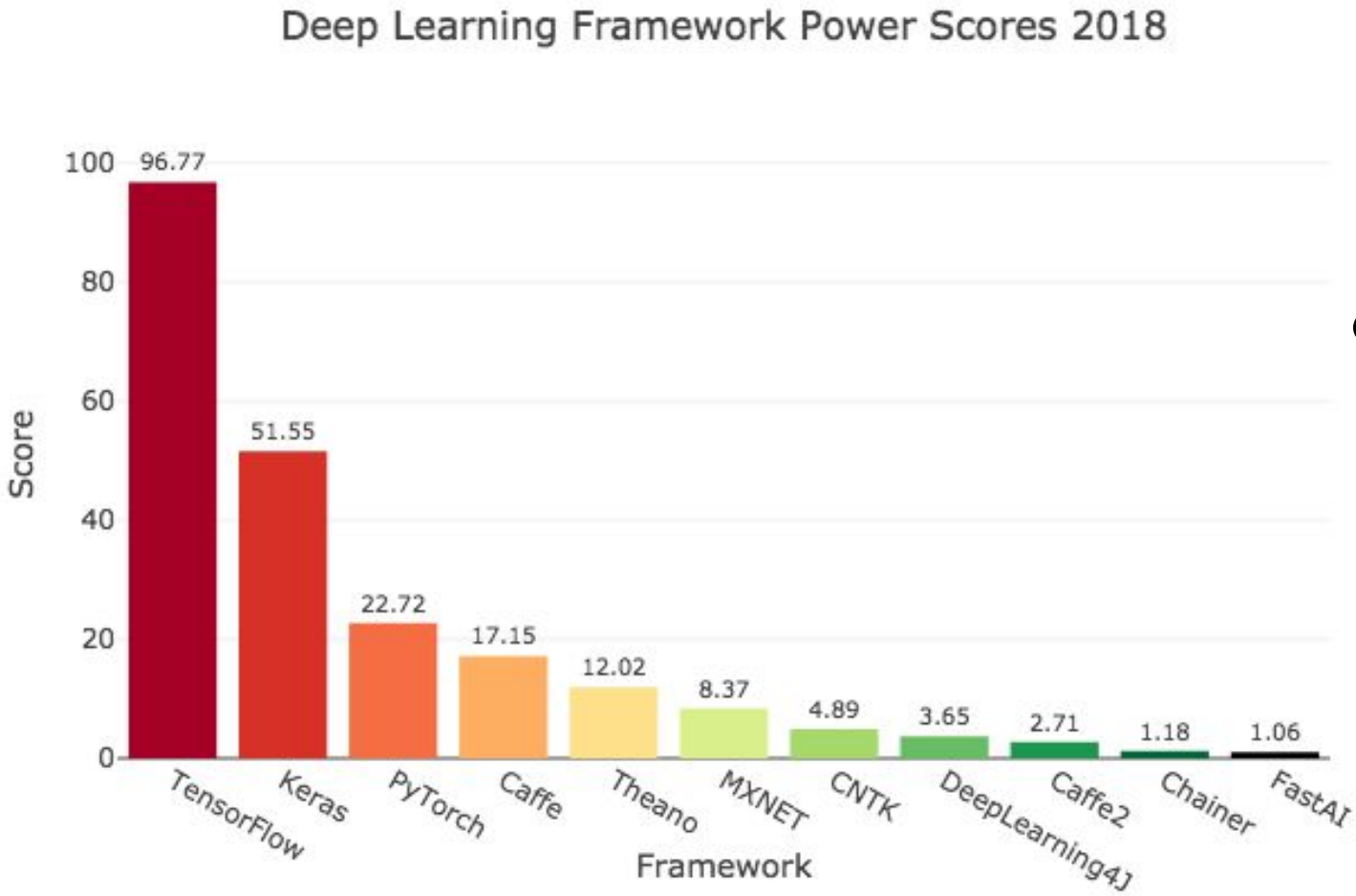
```
Using TensorFlow backend.
```

TensorFlow - Nueva API **Eager**

- Nueva API permite ejecución **Eager**
  - Imperativa
  - Se ejecutan inmediatamente
  - Keras NO las utiliza
- Ejemplo con API eager

```
# habilitar API eager
tf.enable_eager_execution()


x = INPUT # un vector de numpy
W = tf.Variable(np.random.randn())
b = tf.Variable(np.random.randn())

# Salida
C = tf.nn.relu(tf.add(tf.multiply(x, W), b))
# C ya tiene el output (vector de numpy)
```

TensorFlow vs Keras vs Otros (2018)



- En este curso sólo usamos Keras
  - Simple
  - Fácil de aprender
- El interop entre tf y keras es muy bueno
  - Podés combinar ambos en un mismo proyecto
  - Usar tf solo donde es necesario

Resumen

Keras

- Alto nivel (capas)
- Usa TF como backend
  - Backend: provee la implementación real de las operaciones
- Fácil uso
- Input y output con arreglos de numpy
- Puede usar operadores/capas custom implementadas en TF

Tensorflow (TF)

- Nivel intermedio (tensores)
  - tensores = matrices n-dim
- Muy customizable
- Útil para definir nuevos operadores
- tf.tensor: clase propia de tensor
  - convertir desde/hacia numpy
- Repetitivo para redes
  - Keras está incluido en TF para facilitar su uso
- Entrenamiento y predicción “manuales”