TensorFlow Tutorial

Laura Manduchi, Dario Pavllo

ETH Zürich

11–12 April 2019

What is TensorFlow?

- TensorFlow is a deep learning framework for Python
- Created and maintained by Google
- Latest version: 2.0 alpha (stable: 1.13)
- PyTorch (by Facebook) is its major competitor

What does it do?

Like NumPy, TensorFlow is a tensor (N-d array) library.

In addition, it features:

- GPU support and distributed computing
- Building blocks for ML and neural networks
  - Architectures, layers, losses, optimizers, etc.
- **Automatic differentiation**

Computational graph

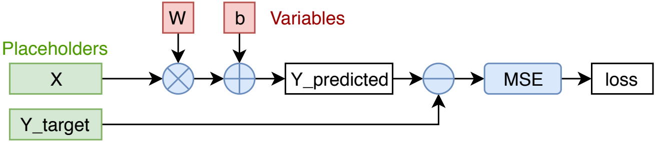
Unlike NumPy, which has an imperative interface, TensorFlow requires defining a static *computational graph*.

- This allows performance optimizations on the graph
- After constructing the graph, the actual computations must be wrapped in a `tf.Session`
- Drawback: hard to inspect intermediate results

Computational graph

- Dynamic inputs: `tf.placeholder`
  - Populated at runtime (i.e. in a `tf.Session`) by means of a *feed dictionary* (`feed_dict`)
- Learnable parameters: `tf.Variable`
  - Must be initialized (usually randomly)
- Operations

Example – linear regression MSE: $\frac{1}{N}\sum_n \|\mathbf{W}\,\mathbf{x_n} + \mathbf{b} - \mathbf{y_n}\|^2$



TensorFlow evaluation vs NumPy

Same example: linear regression MSE ($\mathbb{R}^{10} \to \mathbb{R}^3$)

```python
X_ph = tf.placeholder(tf.float32, [None, 10])
Y_ph = tf.placeholder(tf.float32, [None, 3])

W = tf.Variable(tf.random.normal([10, 3], stddev=1))
b = tf.Variable(tf.random.normal([3], stddev=1))

Y_predicted = tf.matmul(X_ph, W) + b
loss = tf.reduce_mean(tf.reduce_sum((Y_predicted - Y_ph)**2, axis=1))

with tf.Session() as session:
    session.run(tf.global_variables_initializer())
    loss_value, = session.run([loss], feed_dict={X_ph: X, Y_ph: Y})
```

TensorFlow

```python
W = np.random.normal(size=(10, 3))
b = np.random.normal(size=(3,))
Y_predicted = np.dot(X, W) + b
loss = np.mean(np.sum((Y_predicted - Y)**2, axis=1))
```

NumPy

Automatic differentiation

TensorFlow can compute gradients on an arbitrary graph, with respect to arbitrary tensors (`tf.Variable`).

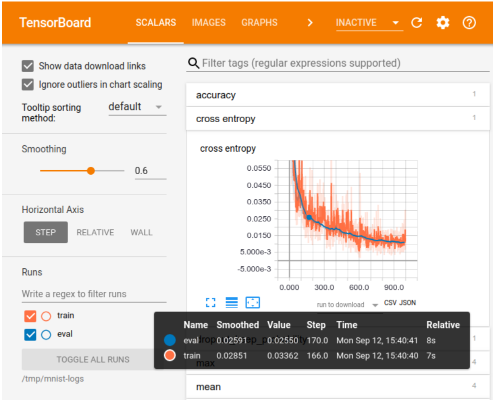
Typical workflow:

1. Define model architecture
2. Define loss function (MSE, MAE, cross-entropy, etc.)
3. Choose an optimizer (SGD, Adam, Rmsprop, etc.)
4. Run optimizer in a `tf.Session`

```python
X_ph = tf.placeholder(tf.float32, [None, 10])
Y_ph = tf.placeholder(tf.float32, [None, 3])

W = tf.Variable(tf.random.normal([10, 3], stddev=1))
b = tf.Variable(tf.random.normal([3], stddev=1))

Y_predicted = tf.matmul(X_ph, W) + b
loss = tf.reduce_mean(tf.reduce_sum((Y_predicted - Y_ph)**2, axis=1))

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
train_op = optimizer.minimize(loss)

with tf.Session() as session:
    session.run(tf.global_variables_initializer())
    for i in range(100):
        _, = session.run([train_op], feed_dict={X_ph: X, Y_ph: Y})
```

TensorBoard

Toolkit for inspection of results and training curves
Command: `tensorboard --logdir=path/to/log-directory`



`https://www.tensorflow.org/guide/summaries_and_tensorboard`

Keras

- Formerly a wrapper for TensorFlow
- Now integrated within TensorFlow
- Provides a high-level and easy-to-use interface
- Useful for quick experiments, but the low-level TensorFlow API is still important for custom architectures

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

PyTorch

- Evolution of Lua Torch
- Maintained by Facebook
- PyTorch has a NumPy-like imperative interface
- The content of tensors can be inspected on the fly
- Similar functionalities as TensorFlow

TensorFlow vs PyTorch

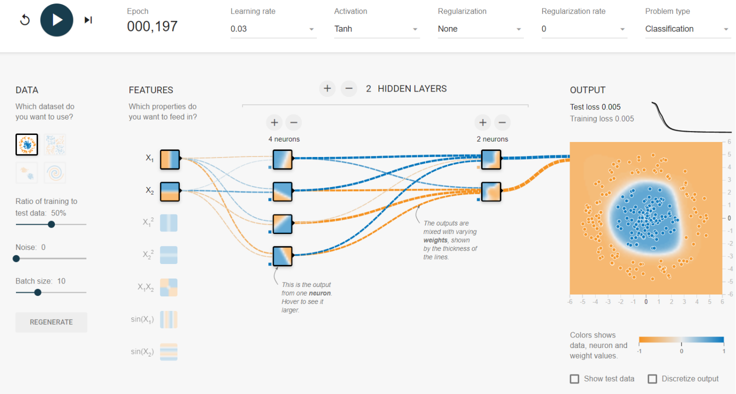
- TensorFlow: construct graph statically, then evaluate it
  - Tedious to use, but allows performance optimizations and compilation in GPU assembly
  - TF has recently introduced an imperative interface (*eager execution*), but it is a separate feature
- PyTorch: graph constructed dynamically during runtime
  - Easier to use/debug
  - Each operator launches a precompiled CUDA kernel
  - Enables some fancy architectures (Dynamic RNNs)
- In theory, TensorFlow should be faster
  - For most uses, performance is the same as they call the same low-level API (CuDNN)

Linear regression in PyTorch

```
W = nn.Parameter(torch.randn(10, 3))
b = nn.Parameter(torch.randn(3))

optimizer = optim.SGD([W, b], lr=0.1)

for i in range(100):
    optimizer.zero_grad()
    Y_predicted = torch.matmul(torch.from_numpy(X), W) + b
    loss = torch.mean(torch.sum((Y_predicted - torch.from_numpy(Y))**2, dim=1))
    loss.backward()
    optimizer.step()
```

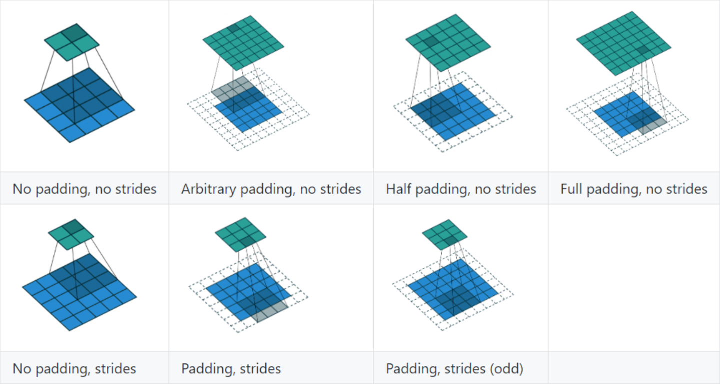
Further Resources

TensorFlow Playground



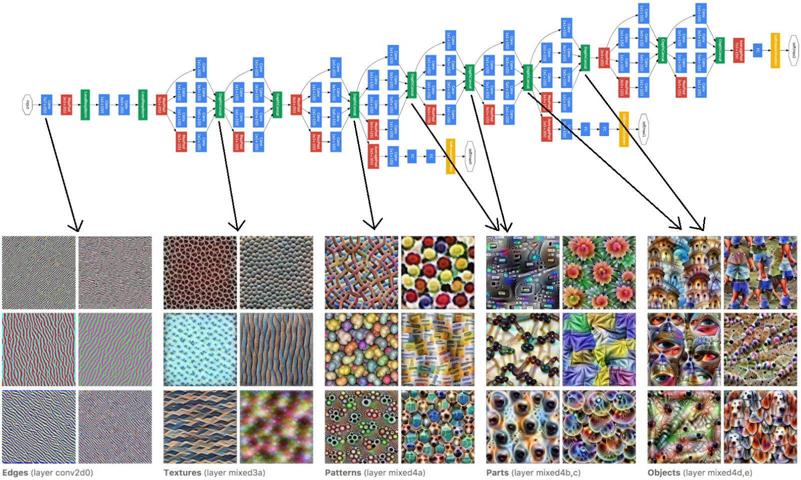
https://playground.tensorflow.org

Convolution Demo



`https://github.com/vdumoulin/conv_arithmetic`

Feature Visualization



https://distill.pub/2017/feature-visualization/