CONTINUING EDUCATION CENTRE

Data Analytics and Machine Learning Using Python

Deep Learning Using Keras-Tensorflow

CONTINUING EDUCATION CENTRE

Outline

- Tensorflow concept
- Constructing models using Keras for Tensorflow
- Example: CNN and LSTM

CONTINUING EDUCATION CENTRE

TF-Intro

- TensorFlow (TF) is an open-source machine learning library for research and production.
  - TF offers APIs for beginners and experts to develop for desktop, mobile, web, and cloud. See the sections below to get started.
  - Currently, it is the most popular deep learning library.
- TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.
- TensorFlow computations are expressed as stateful dataflow graphs.
  - The name derives from the operations that such neural networks perform on multidimensional data arrays (a.k.a tensors).

CONTINUING EDUCATION CENTRE

TF-Intro

- The fundamental TensorFlow Core programs consist of two discrete steps:
  1. Building the computational graph (a tf.Graph).
  2. Running the computational graph (using a tf.Session).
- On top of those two fundamental elements, we can build a complex deep learning framework that consists high level components such as datasets, layers, and feature_columns.
- Normally, TF computations run within a loop that are controlled by the so-called Estimators.

CONTINUING EDUCATION CENTRE

TF-Intro::a Glimpse Look

```
x = tf.constant([[1], [2], [3], [4]], dtype=tf.float32)
y_true = tf.constant([[0], [-1], [-2], [-3]], dtype=tf.float32)

linear_model = tf.layers.Dense(units=1)
y_pred = linear_model(x)
loss = tf.losses.mean_squared_error(labels=y_true, predictions=y_pred)

optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()

sess = tf.Session()
sess.run(init)
for i in range(100):
  _, loss_value = sess.run((train, loss))
  print(loss_value)

print(sess.run(y_pred))
```

CONTINUING EDUCATION CENTRE

TF-Core::Graph

- A **computational graph** is a series of TensorFlow operations arranged into a graph. The graph is composed of two types of objects:
  - **tf.Operation**: The nodes of the graph. Operations describe calculations that consume and produce tensors.
  - **tf.Tensor**: The edges in the graph. These represent the values that will flow through the graph. Most TensorFlow functions return tf.Tensors

Remark: tf.Tensors do not have values, they are just handles to elements in the computation graph.

TF-Core::Graph

Example:

```
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
print(a)
print(b)
print(total)
```

Output:

```
Tensor("Const:0", shape=(), dtype=float32)
Tensor("Const_1:0", shape=(), dtype=float32)
Tensor("add:0", shape=(), dtype=float32)
```

CONTINUING EDUCATION CENTRE

TF-Core::Graph

TensorBoard

TensorFlow provides a utility called TensorBoard for visualizing a computation graph:

```
writer = tf.summary.FileWriter('.')
writer.add_graph(tf.get_default_graph())
writer.flush()
```

Tensorboard can be used to display the graph using:

```
tensorboard --logdir .
```

This will produce graph such as:

CONTINUING EDUCATION CENTRE

TF-Core::Session

- A session encapsulates the state of the TensorFlow runtime, and runs TensorFlow operations.
- A tf.Session object can be used to evaluate tensors.

```
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
print(a)
print(b)
print(total)
sess = tf.Session()
print(sess.run(total))    # This prints the expected value of 7.0:
```

CONTINUING EDUCATION CENTRE

TF-Core::Session

- During a call to tf.Session.run any tf.Tensor only has a single value.
- For example, the following code calls tf.random_uniform to produce a tf.Tensor that generates a random 3-element vector (with values in [0,1)):

```
vec = tf.random_uniform(shape=(3,))
out1 = vec + 1
out2 = vec + 2
sess = tf.Session()
print(sess.run(vec))
print(sess.run(vec))
print(sess.run((out1, out2)))
```

CONTINUING EDUCATION CENTRE

TF-Core::Session

- The result shows a different random value on each call to run, but a consistent value during a single run (out1 and out2 receive the same random input):

```
[ 0.52917576  0.64076328  0.68353939]
[ 0.66192627  0.89126778  0.06254101]
(
  array([ 1.88408756,  1.87149239,  1.84057522], dtype=float32),
  array([ 2.88408756,  2.87149239,  2.84057522], dtype=float32)
)
```

CONTINUING EDUCATION CENTRE

TF-Core::Layers

- Layers (tf.layers) are the preferred way to add trainable parameters to a graph.
- For example a densely-connected layer performs a weighted sum across all inputs for each output and applies an optional activation function. The connection weights and biases are managed by the layer object.

```
x = tf.placeholder(tf.float32, shape=[None, 3])
linear_model = tf.layers.Dense(units=1)
y = linear_model(x)
```

CONTINUING EDUCATION CENTRE

TF-Core::Layers

- The layer contains variables that must be initialized before they can be used. For example:

```
init = tf.global_variables_initializer() #jcreate a handle, not running yet
sess.run(init)                #actually run the operation
```

- Now that the layer is initialized, we can evaluate the linear_model's output tensor as we would any other tensor. For example, the following code:

```
print(sess.run(y, {x: [[1, 2, 3],[4, 5, 6]]}))
```

will generate a two-element output vector:

```
[[-3.41378999]
 [-9.14999008]]
```

CONTINUING EDUCATION CENTRE

TF-Core::Training

- The training involves several steps.
  First let's define some inputs, x, and the expected output for each input, y_true:

```
x = tf.constant([[1], [2], [3], [4]], dtype=tf.float32)
y_true = tf.constant([[0], [-1], [-2], [-3]], dtype=tf.float32)
```

- Next, build a simple linear model, with 1 output:

```
linear_model = tf.layers.Dense(units=1)
y_pred = linear_model(x)
```

CONTINUING EDUCATION CENTRE

TF-Core::Training

- The prediction can be evaluated as follows:

```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

print(sess.run(y_pred))
```

- The model hasn't yet been trained, so the four "predicted" values aren't very good:

```
[[ 0.02631879]
 [ 0.05263758]
 [ 0.07895637]
 [ 0.10527515]]
```

CONTINUING EDUCATION CENTRE

TF-Core::Training

- To optimize a model, we first need to define the loss. We'll use the mean square error, a standard loss for regression problems.

- The tf.losses module provides a set of common loss functions. We can use it to calculate the mean square error as follows:

```
loss = tf.losses.mean_squared_error(labels=y_true, predictions=y_pred)

print(sess.run(loss))
```

This will produce a loss value, something like:

```
2.23962
```

CONTINUING EDUCATION CENTRE

TF-Core::Training

- TensorFlow provides optimizers implementing standard optimization algorithms. These are implemented as sub-classes of tf.train.Optimizer.
- The simplest optimization algorithm is gradient descent, implemented by tf.train.GradientDescentOptimizer. It modifies each variable according to the magnitude of the derivative of loss with respect to that variable. For example:

```
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
```

CONTINUING EDUCATION CENTRE

TF-Core::Training

- The following code builds all the graph components necessary for the optimization, and returns a training operation. When run, the training op will update variables in the graph:

```
for i in range(100):
  _, loss_value = sess.run((train, loss))
  print(loss_value)
```

print (list_value) will show the progression of the loss during training:

```
1.35659
1.00412
0.759167
...
```

CONTINUING EDUCATION CENTRE

TF-Core::Putting them altogether

```
x = tf.constant([[1], [2], [3], [4]], dtype=tf.float32)
y_true = tf.constant([[0], [-1], [-2], [-3]], dtype=tf.float32)

linear_model = tf.layers.Dense(units=1)
y_pred = linear_model(x)
loss = tf.losses.mean_squared_error(labels=y_true, predictions=y_pred)

optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
for i in range(100):
  _, loss_value = sess.run((train, loss))
  print(loss_value)

print(sess.run(y_pred))
```

CONTINUING EDUCATION CENTRE

TF-Core::More info

Tensorflow has several more mechanism to handle data, which includes Variable, Placeholder, Dataset and Feature Column.

CONTINUING EDUCATION CENTRE

TF-Core::Variable

- A TensorFlow variable is the best way to represent shared, persistent state manipulated by your program.
- A tf.Variable represents a tensor whose value can be changed by running ops on it. Unlike tf.Tensor objects, a tf.Variable exists outside the context of a single session.run call. These modifications are visible across multiple tf.Sessions.
- The best way to create a variable is to call the tf.get_variable function:

```
my_variable = tf.get_variable("my_variable", [1, 2, 3])
```

  This creates a variable named "my_variable" which is a three-dimensional tensor with shape [1, 2, 3].
- Before you can use a variable, it must be initialized.
  To initialize all trainable variables in one go, before training starts, call tf.global_variables_initializer().

CONTINUING EDUCATION CENTRE

TF-Core::Placeholders

- A graph can be parameterized to accept external inputs, known as placeholders. A placeholder is a promise to provide a value later, like a function argument.

```
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = x + y
sess = tf.Session()
print(sess.run(z, feed_dict={x: 3, y: 4.5}))
print(sess.run(z, feed_dict={x: [1, 3], y: [2, 4]}))
```

This results in the following output:

```
7.5
[ 3.  7.]
```

CONTINUING EDUCATION CENTRE

TF-Core::Placeholders vs Variable

- We use tf.Variable for trainable variables such as weights (W) and biases (B) for your model.

```
weights = tf.Variable(
 tf.truncated_normal([IMAGE_PIXELS, hidden1_units],
                     stddev=1.0 / math.sqrt(float(IMAGE_PIXELS))), name='weights')

biases = tf.Variable(tf.zeros([hidden1_units]), name='biases')
```

- tf.placeholder is used to feed actual training examples.

```
images_placeholder = tf.placeholder(tf.float32, shape=(batch_size,
IMAGE_PIXELS))
labels_placeholder = tf.placeholder(tf.int32, shape=(batch_size))
```

CONTINUING EDUCATION CENTRE

TF-Core::Placeholders vs Variable

- To use the placeholders to feed the training examples during the training:

```
for step in xrange(FLAGS.max_steps):
    feed_dict = {
      images_placeholder: images_feed,
      labels_placeholder: labels_feed,
     }
    _, loss_value = sess.run([train_op, loss], feed_dict=feed_dict)
```

tf.variables objects will be trained (modified) as the result of this training.

CONTINUING EDUCATION CENTRE

TF-Core::Datasets

- Placeholders work for simple experiments, but tf.data are the preferred method of streaming data into a model.

- To get a runnable tf.Tensor from a Dataset we must first convert it to a tf.data.Iterator, and then call the Iterator's tf.data.Iterator.get_next method.

- The simplest way to create an Iterator is with the tf.data.Dataset.make_one_shot_iterator method.

TF-Core::Datasets

- If the Dataset depends on stateful operations we may need to initialize the iterator before using it, as shown below:

```
r = tf.random_normal([10,3])
dataset = tf.data.Dataset.from_tensor_slices(r)
iterator = dataset.make_initializable_iterator()
next_row = iterator.get_next()

sess.run(iterator.initializer)
while True:
  try:
    print(sess.run(next_row))
  except tf.errors.OutOfRangeError:
    break
```

CONTINUING EDUCATION CENTRE

TF-Core::Features

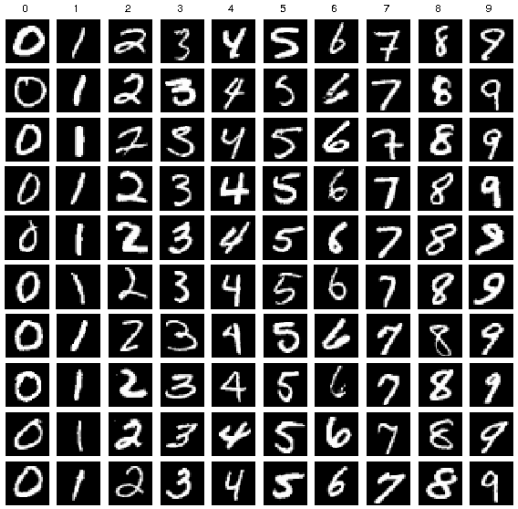
- A nice thing to work with Tensorflow is that we can provide inputs with special format, called feature column.

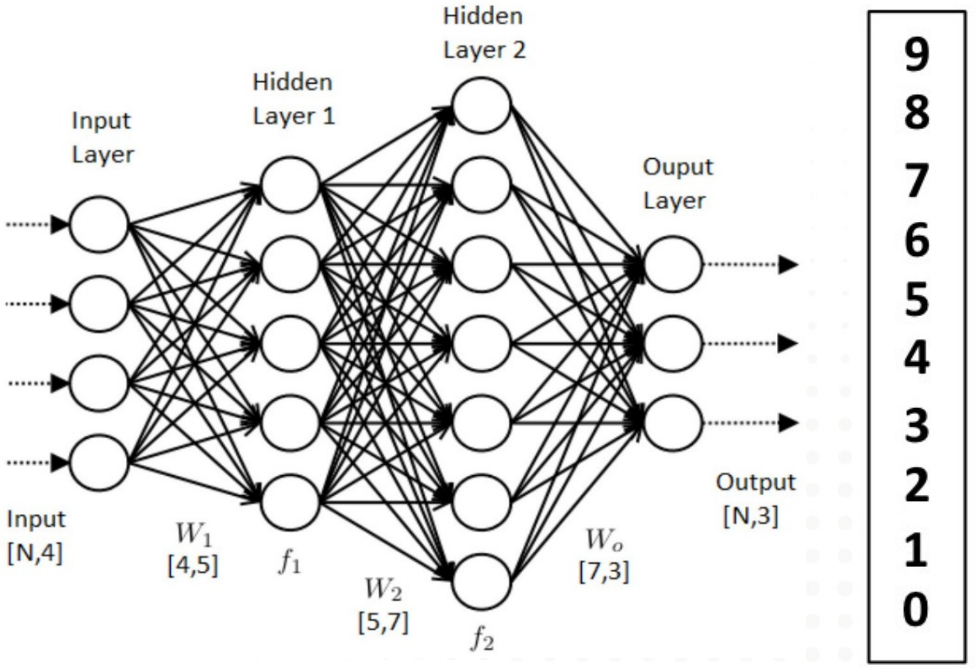
- The easiest way to experiment with feature columns is using the tf.feature_column.input_layer function. This function only accepts dense columns as inputs, so to view the result of a categorical column you must wrap it in an tf.feature_column.indicator_column.

- Feature columns can have internal state, like layers, so they often need to be initialized.

CONTINUING EDUCATION CENTRE

More Complex Example





9
8
7
6
5
4
3
2
1
0

See: 1_ann_without_keras.py

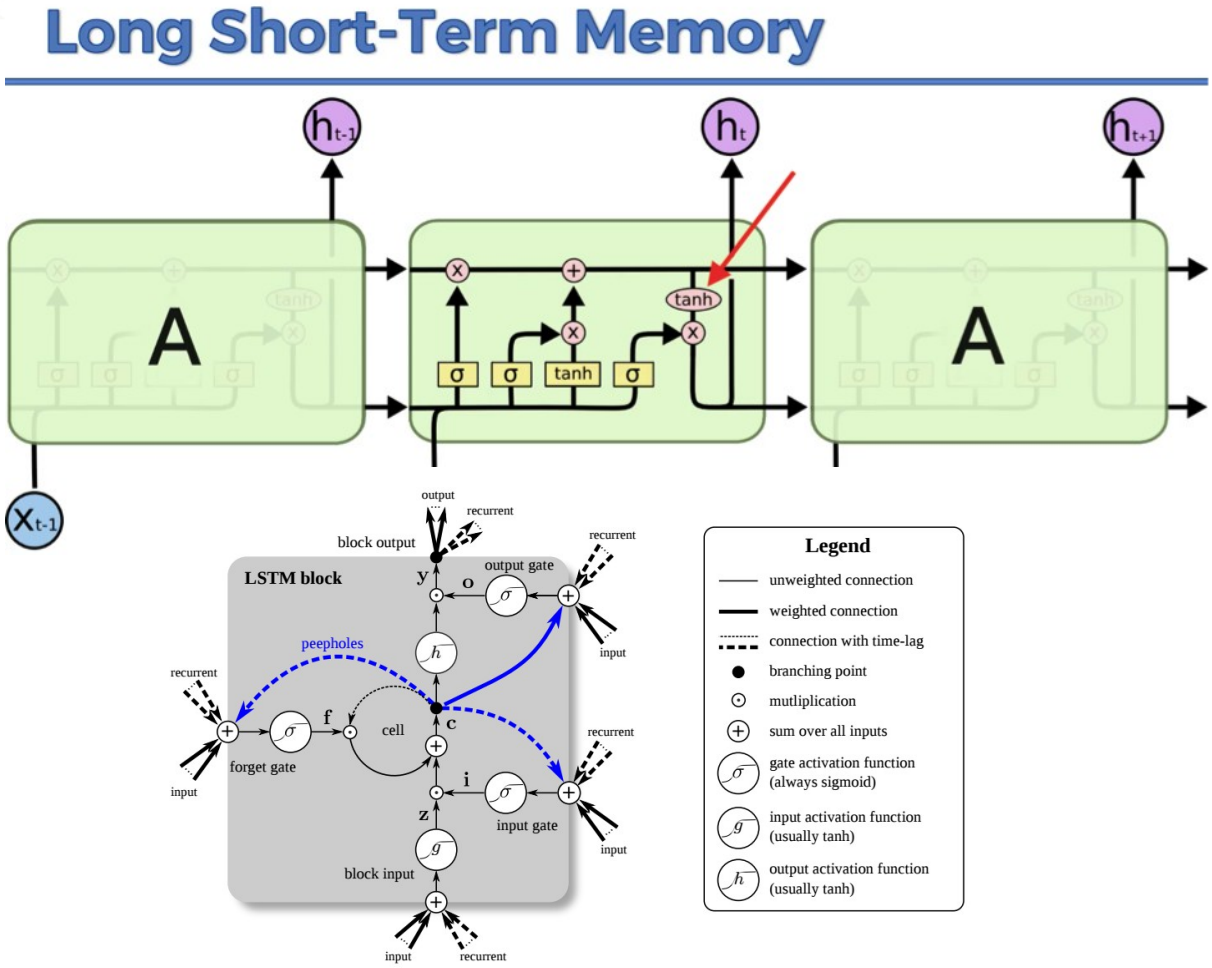
CONTINUING EDUCATION CENTRE

More Complex Example

Time Series Analysis

- Long short-term memory (LSTM) is another example of neural network with deep structure.
- LSTM units are units of a recurrent neural network (RNN). An RNN composed of LSTM units is often called an LSTM network. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate.

More Complex Example::LSTM



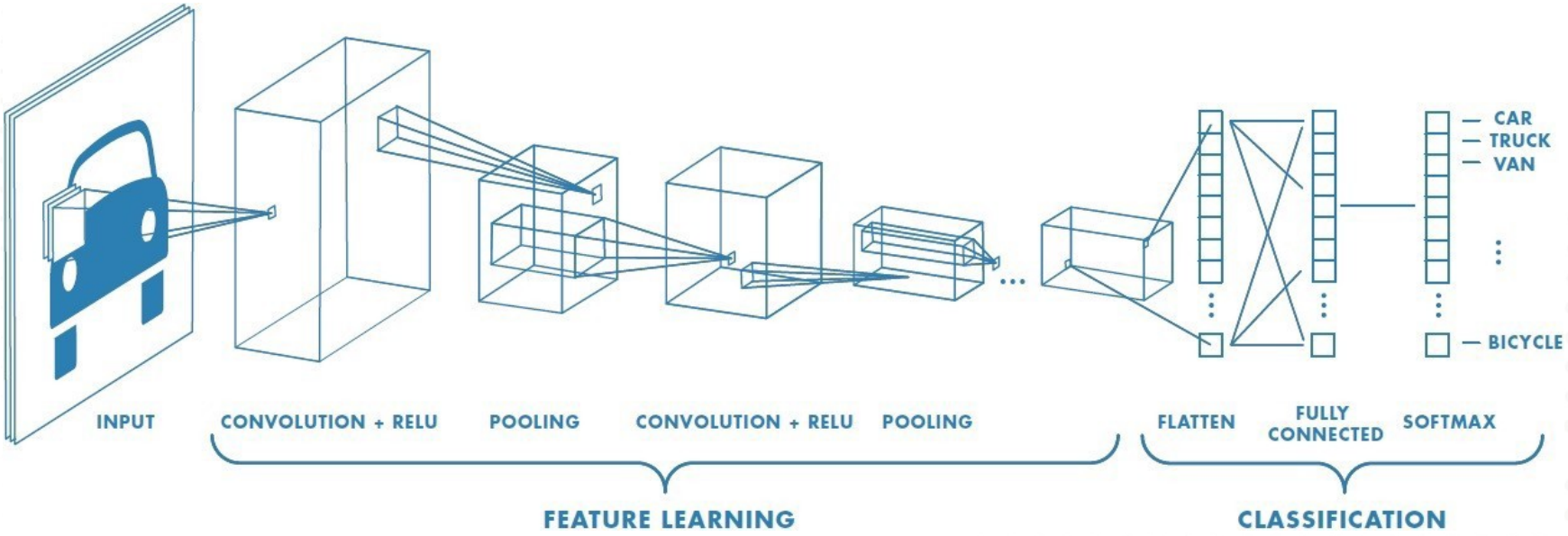
See: 2_lstm_stock_market_prediction.py

CONTINUING EDUCATION CENTRE

More Complex Example::CNN

Image Recognition

Convolutional neural networks (CNN) are the state of the art technique for image recognition.



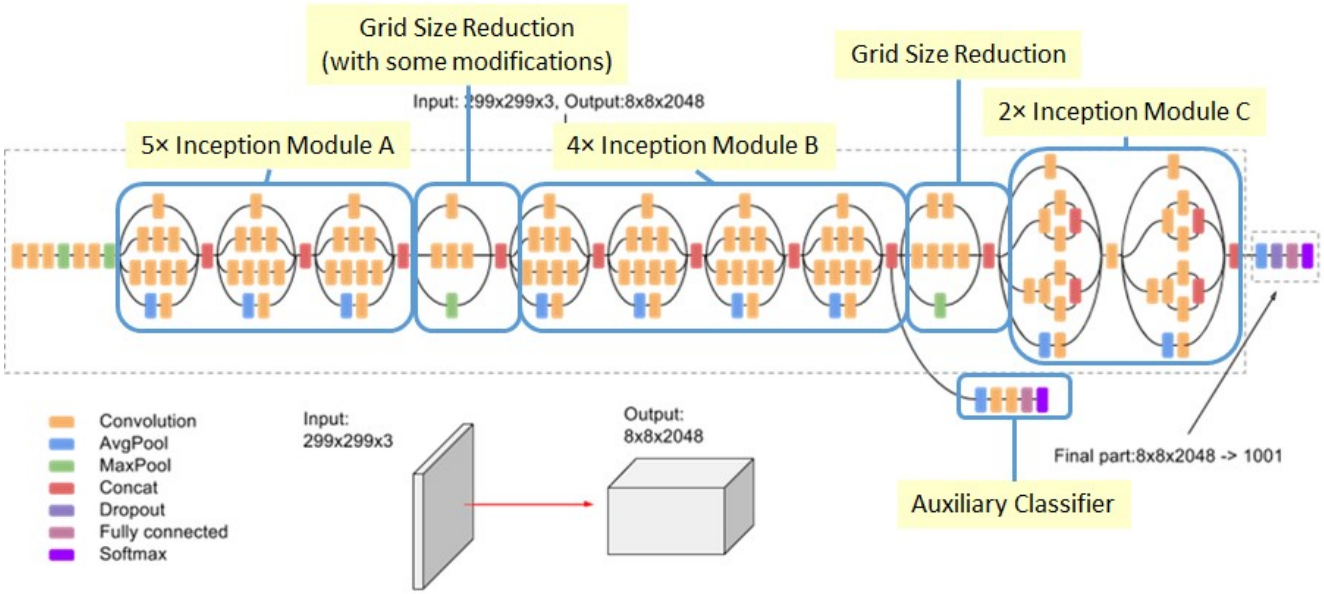
CONTINUING EDUCATION CENTRE

More Complex Example

Google Inception

Convolutional neural networks (CNN) are the state of the art technique for image recognition.

Google has a number of neural network models that they have made available for use in TensorFlow.

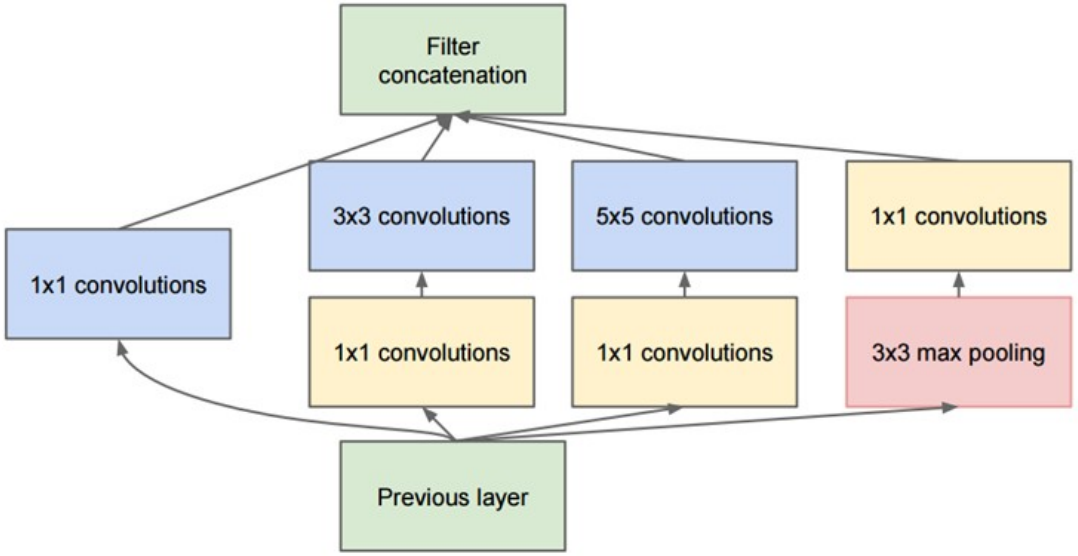


CONTINUING EDUCATION CENTRE

More Complex Example

Google Inception

The core lays in the inception modules which take several convolutional kernels of different sizes and stack their outputs along the depth dimension in order to capture features at different scales.



Full Inception module

See: classify_image.py

CONTINUING EDUCATION CENTRE

Keras-Intro

- Developing a machine learning program natively using Tensorflow might be cumbersome.
- Keras is a minimalist Python library for deep learning that can run on top of Theano or TensorFlow.
- It was developed by François Chollet (a Google engineer) to make implementing deep learning models as fast and easy as possible for research and development.
- It runs on Python 2.7 or 3.5 and can seamlessly execute on GPUs and CPUs given the underlying frameworks. It is released under the permissive MIT license.

CONTINUING EDUCATION CENTRE

Keras-Intro

Here is the keras version for the previous SLP (Single Layer Perceptron)

```
import tensorflow as tf

x = tf.constant([[1], [2], [3], [4]], dtype=tf.float32)
y_true = tf.constant([[0], [-1], [-2], [-3]], dtype=tf.float32)

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(1)])

model.compile(optimizer='sgd',
              loss='mean_squared_error')

model.fit(x, y_true, epochs=100,steps_per_epoch=1)
x_test = tf.constant([[5], [6], [3], [4]], dtype=tf.float32)
model.predict(x_test)
```

CONTINUING EDUCATION CENTRE

Keras-Intro::a Glimpse Look

```
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(),
  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)
```

CONTINUING EDUCATION CENTRE

LSTM Using Keras-TF

See:

- 3_lstm_data_airline.py
- 4_lstm_air_pollution_multivariate.py

CONTINUING EDUCATION CENTRE

References

- https://www.tensorflow.org/tutorials/
- https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#0
- https://www.datacamp.com/community/tutorials/tensorflow-tutorial
- https://www.guru99.com/tensorflow-tutorial.html
- https://www.datacamp.com/community/tutorials/deep-learning-python
- https://machinelearningmastery.com/introduction-python-deep-learning-library-keras/