package com.twitter.search.common.util.ml.tensorflow\_engine;

import java.io.IOException;

import java.util.Collections;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Supplier;

import com.google.common.base.Preconditions;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.tensorflow.SavedModelBundle;

import org.tensorflow.Session;

import com.twitter.ml.api.FeatureUtil;

import com.twitter.search.common.features.thrift.ThriftSearchFeatureSchema;

import com.twitter.search.common.features.thrift.ThriftSearchFeatureSchemaEntry;

import com.twitter.search.common.file.AbstractFile;

import com.twitter.search.common.schema.DynamicSchema;

import com.twitter.search.common.util.ml.models\_manager.BaseModelsManager;

import com.twitter.tfcompute\_java.TFModelRunner;

import com.twitter.tfcompute\_java.TFSessionInit;

import com.twitter.twml.runtime.lib.TwmlLoader;

import com.twitter.twml.runtime.models.ModelLocator;

import com.twitter.twml.runtime.models.ModelLocator$;

import com.twitter.util.Await;

/\*\*

\* TensorflowModelsManager manages the lifecyle of TF models.

\*/

public class TensorflowModelsManager extends BaseModelsManager<TFModelRunner> {

private static final Logger LOG = LoggerFactory.getLogger(TensorflowModelsManager.class);

private static final String[] TF\_TAGS = new String[] {"serve"};

private volatile Map<Integer, Long> featureSchemaIdToMlApiId = new HashMap<Integer, Long>();

static {

TwmlLoader.load();

}

public static final TensorflowModelsManager NO\_OP\_MANAGER =

createNoOp("no\_op\_manager");

public TensorflowModelsManager(

Supplier<Map<String, AbstractFile>> activeModelsSupplier,

boolean shouldUnloadInactiveModels,

String statsPrefix

) {

this(

activeModelsSupplier,

shouldUnloadInactiveModels,

statsPrefix,

() -> true,

() -> true,

null

);

}

public TensorflowModelsManager(

Supplier<Map<String, AbstractFile>> activeModelsSupplier,

boolean shouldUnloadInactiveModels,

String statsPrefix,

Supplier<Boolean> serveModels,

Supplier<Boolean> loadModels,

DynamicSchema dynamicSchema

) {

super(

activeModelsSupplier,

shouldUnloadInactiveModels,

statsPrefix,

serveModels,

loadModels

);

if (dynamicSchema != null) {

updateFeatureSchemaIdToMlIdMap(dynamicSchema.getSearchFeatureSchema());

}

}

/\*\*

\* The ML API feature ids for tensorflow scoring are hashes of their feature names. This hashing

\* could be expensive to do for every search request. Instead, allow the map from schema feature

\* id to ML API id to be updated whenever the schema is reloaded.

\*/

public void updateFeatureSchemaIdToMlIdMap(ThriftSearchFeatureSchema schema) {

HashMap<Integer, Long> newFeatureSchemaIdToMlApiId = new HashMap<Integer, Long>();

Map<Integer, ThriftSearchFeatureSchemaEntry> featureEntries = schema.getEntries();

for (Map.Entry<Integer, ThriftSearchFeatureSchemaEntry> entry : featureEntries.entrySet()) {

long mlApiFeatureId = FeatureUtil.featureIdForName(entry.getValue().getFeatureName());

newFeatureSchemaIdToMlApiId.put(entry.getKey(), mlApiFeatureId);

}

featureSchemaIdToMlApiId = newFeatureSchemaIdToMlApiId;

}

public Map<Integer, Long> getFeatureSchemaIdToMlApiId() {

return featureSchemaIdToMlApiId;

}

/\*\*

\* If the manager is not enabled, it won't fetch TF models.

\*/

public boolean isEnabled() {

return true;

}

/\*\*

\* Load an individual model and make it available for inference.

\*/

public TFModelRunner readModelFromDirectory(

AbstractFile modelDir) throws IOException {

ModelLocator modelLocator =

ModelLocator$.MODULE$.apply(

modelDir.toString(),

modelDir.toURI()

);

try {

Await.result(modelLocator.ensureLocalPresent(true));

} catch (Exception e) {

LOG.error("Couldn't find model " + modelDir.toString(), e);

throw new IOException("Couldn't find model " + modelDir.toString());

}

Session session = SavedModelBundle.load(modelLocator.localPath(), TF\_TAGS).session();

return new TFModelRunner(session);

}

/\*\*

\* Initialize Tensorflow intra and inter op thread pools.

\* See `ConfigProto.[intra|inter]\_op\_parallelism\_threads` documentation for more information:

\* https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/protobuf/config.proto

\* Initialization should happen only once.

\* Default values for Tensorflow are:

\* intraOpParallelismThreads = 0 which means that TF will pick an appropriate default.

\* interOpParallelismThreads = 0 which means that TF will pick an appropriate default.

\* operation\_timeout\_in\_ms = 0 which means that no timeout will be applied.

\*/

public static void initTensorflowThreadPools(

int intraOpParallelismThreads,

int interOpParallelismThreads) {

new TFSessionInit(intraOpParallelismThreads, interOpParallelismThreads, 0);

}

/\*\*

\* Creates a no-op instance. It can be used for tests or when the models are disabled.

\*/

public static TensorflowModelsManager createNoOp(String statsPrefix) {

return new TensorflowModelsManager(Collections::emptyMap, false, statsPrefix) {

@Override

public void run() { }

@Override

public boolean isEnabled() {

return false;

}

@Override

public void updateFeatureSchemaIdToMlIdMap(ThriftSearchFeatureSchema schema) { }

};

}

/\*\*

\* Creates an instance that loads the models based on a ConfigSupplier.

\*/

public static TensorflowModelsManager createUsingConfigFile(

AbstractFile configFile,

boolean shouldUnloadInactiveModels,

String statsPrefix,

Supplier<Boolean> serveModels,

Supplier<Boolean> loadModels,

DynamicSchema dynamicSchema) {

Preconditions.checkArgument(

configFile.canRead(), "Config file is not readable: %s", configFile.getPath());

return new TensorflowModelsManager(

new ConfigSupplier(configFile),

shouldUnloadInactiveModels,

statsPrefix,

serveModels,

loadModels,

dynamicSchema

);

}

}