VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

Tensorflow 2.0 and Keras

Georg Zitzlsberger georg.zitzlsberger@vsb.cz

26-03-2021

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research, Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS

Agenda

Introduction

Data Pipeline

Data Pipeline: Exercise

Building a Model

Building a Model: Exercise

Train and Visualize with Tensorboard

Train and Visualize with Tensorboard: Exercise

Tensorflow Dataset Recommendations

Introduction

- Tensorflow 2.0 introduced end of 2019
- Includes Keras now
- Latest version: 2.4.1
- APIs for C++, Java and **Python**
- The "biggest" community, but also lot's of changes
- Applicable to a wide range of user types:
  - developer
  - researcher
  - industry or academia
- Enhanced versions are available for different platforms, e.g. via PIP:
  - Intel CPUs `intel-tensorflow`
  - AMD CPUs `tensorflow-rocm`
  - NVIDIA `tensorflow-gpu`

TensorFlow

Keras

Data Pipeline

- Extract, Transform and Load (ETL) pipeline via `tf.data.Dataset`
- Provides a wide range of functionality to process training/validation data:
  - I/O: files, NumPy, TFRecord/Protocol Buffers, Pandas Data Frames, etc.
  - Split training/validation:
    Provide a ratio how much of the dataset should be for training.
  - Batch and pad:
    Build minibatches and pad to ensure balance.
  - Shuffle:
    Randomize the samples with every training epoch.
  - Cache and Pre-fetch:
    Optimize access to data.
  - Map and filter:
    Convert the data to a format needed for training/validation and also filter samples.
  - ...

Data Pipeline: Exercise

- In the exercise `keras_mnist_example_1.ipynb`, build a training pipeline with:
  - Input MNIST training dataset `ds_train` and apply `normalize_img` with `tf.data.experimental.AUTOTUNE` parallel calls, via
    ▸ `tf.data.Dataset.map`
  - Cache the data (no repeated normalization) with
    ▸ `tf.data.Dataset.cache`
  - Suffle data entirely (size of `ds_info.splits['train'].num_examples`) with
    ▸ `tf.data.Dataset.shuffle`
  - Batch with a batch size of 128 with
    ▸ `tf.data.Dataset.batch`
  - Prefetch the next elements (use `tf.data.experimental.AUTOTUNE` for the buffer size)
    ▸ `tf.data.Dataset.prefetch`
- For the validation pipeline `ds_test`, do the same **except** suffling

Use the ▸ Tensorflow Dataset Documentation to help you

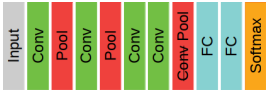
Building a Model

- Keras offers two ways to build a model:
  - Sequential model with `tf.keras.Sequential`
  - Functional API with `tf.keras.Model`
- Most used operations/layers already exist in the Keras API, e.g.:
  - `tf.keras.layers.Conv2D` or `tf.keras.layers.Conv3D`
  - `tf.keras.layers.Dense`
  - `tf.keras.layers.LSTM`
  - ...
- The models expect data in the following formats (`channels_last`):
  - `[batch, spatial_dims..., channels]`, e.g. 2D images:
    `[10, 256, 256, 3]` (10 per batch, 256x256 images with 3 color channels)
  - `[batch, time_step, spatial_dims..., channels]`, e.g. time series of images:
    `[10, 25, 64, 32, 1]` (10 per batch, series of 25 64x32 images with 1 color channel)

Building a Sequential Model

- Very easy to use with least amount of code
- Only for sequential models - no forks or joins!
- Example:

```
from keras import Sequential
from keras.layers import Conv2D, MaxPooling2D,
  BatchNormalization, ZeroPadding2D, Dropout,
  Activation, Flatten, Dense

def alexnet(n_classes=5):
    model = Sequential()
    model.add(Conv2D(64, 11, strides=4))
    model.add(ZeroPadding2D(2))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=3,
                           strides=2))
    model.add(Conv2D(192, 5))
    model.add(ZeroPadding2D(2))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=3,
                           strides=2))
    model.add(Conv2D(384, 3))
    model.add(ZeroPadding2D(1))
    model.add(Activation('relu'))

    model.add(Conv2D(256, 3))
   model.add(ZeroPadding2D(1))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=3,
                           strides=2))
    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(4096,
                    input_shape=(6 * 6 * 256, )))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4096))
    model.add(Activation('relu'))
    model.add(Dense(n_classes))
    model.add(Activation('softmax'))

    return model

if __name__ == '__main__':
    amodel = alexnet(10)
    amodel.summary()
```

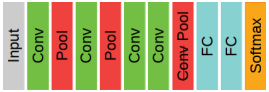


Building a Functional API Model

- Requires definition of input and model (output)
- Allows forks/joins (not shown here)
- Example:

```
from keras import Sequential
from keras.layers import Conv2D, MaxPooling2D,
  BatchNormalization, ZeroPadding2D, Dropout,
  Activation, Flatten, Dense

def alexnet(n_classes=5):
    inp = tf.keras.Input(shape=[256,256,3], dtype=tf.float32)

    conv1 = Conv2D(64, 11, strides=4))(inp)
    pad1 = ZeroPadding2D(2)(conv1)
    act1 = Activation('relu')(pad1)
    pool1 = MaxPooling2D(pool_size=3,
                         strides=2)(act1)
    conv2 = Conv2D(192, 5)(pool1)
    pad2 = ZeroPadding2D(2)(conv2)
    act2 = Activation('relu')(pad2)
    pool2 = MaxPooling2D(pool_size=3,
                         strides=2)(act2)
    conv3 = Conv2D(384, 3)(pool2)
    pad3 = ZeroPadding2D(1)(conv3)
    act3 = Activation('relu')(pad3)

    conv4 = Conv2D(256, 3)(act3)
    pad4 = ZeroPadding2D(1)(conv4)
    act4 = Activation('relu')(pad4)
    pool4 = MaxPooling2D(pool_size=3,
                         strides=2)(act4)
```

```
    flat = Flatten()(pool4)
    drop1 = Dropout(0.5)(flat)
    dense1 = Dense(4096,
                   input_shape=(6 * 6 * 256, ))(drop1)
    act5 = Activation('relu')(dense1)
    drop2 = Dropout(0.5)(act5)
    dense2 = Dense(4096)(drop2)
    act6 = Activation('relu')(dense2)
    dense3 = Dense(n_classes)(act6)
    act7 = Activation('softmax')(dense3)

    return tf.keras.Model(inp, act7)

if __name__ == '__main__':
    amodel = alexnet(10)
    amodel.summary()
```



Building a Model: Exercise

In exercise `keras_mnist_example_2.ipynb` build a simple model with the following layers (from input to output):

- Flatten the 2D input via
  - `tf.keras.layers.Flatten`
- Dense hidden layer with 128 neurons/units and Rectified Linear Unit (ReLU) activation via
  - `tf.keras.layers.Dense`
- Dense layer as output with 10 neurons/units and softmax activation via
  - `tf.keras.layers.Dense`

Train and Visualize with Tensorboard

- Recap - we have:
  - Data pipelines provide training/validation data
  - Model to train
- Select the loss function, e.g.:
  `loss = tf.keras.losses.BinaryCrossentropy()`
- Optionally, select metrics, e.g.:
  `metric1 = tf.keras.metrics.MeanAbsoluteError()`
- Select the optimizer to use, e.g.:
  `opt = tf.keras.optimizers.SGD(lr=.001, momentum=0.8)`
- Compile the model:
  `amodel.compile(optimizer=opt, loss=loss, metrics=[metric1])`

Train and Visualize with Tensorboard

- For Tensorboard, define a callback, e.g.:

```
tensorboard_cb = tf.keras.callbacks.TensorBoard(log_dir="./logs",
                                                histogram_freq=1,
                                                update_freq='batch')
```

- Snapshots needed?

```
save_best_cb = tf.keras.callbacks.ModelCheckpoint(
                                  filepath="./best_weights.hdf5"
                                  monitor='val_loss',
                                  save_best_only=True)
```

- Train...

```
amodel.fit(
    training_ds,
    validation_data = validation_ds,
    epochs = 100,
    callbacks = [save_best_cb, tensorboard_cb],
    verbose=2)
```

Train and Visualize with Tensorboard: Exercise

Train with visualization in Tensorboard using exercise `keras_mnist_example_3.ipynb`:

- Find the epoch loss in Tensorboard in tab *SCALARS*
- Compare the loss with the metric (here: *accuracy*)
- Inspect the model graph in tab *GRAPHS*
- See the change of parameters during training in tab *HISTOGRAMS*

Note: Don't forget to set a reload intervall in settings (or reload manually)!

Tensorflow Dataset Recommendations

- Some methods offer multi-threading; try `tf.data.experimental.AUTOTUNE`, e.g.:

```
train_ds = tf.data.Dataset.from_tensor_slices(my_data)
                    .map(my_prepare_func, num_parallel_calls=AUTO))
```

- Caching is keeping everything in memory - be carefull where to place it in the pipeline!
- Caching can also be used to use fast NVM/SSD storage, e.g.: `train_ds.cache(filename="/mnt/nvmeof/")`
- Use `tf.data.Dataset.map` before `tf.data.Dataset.batch` if map is expensive, vice versa otherwise
- Prefetch at the end of the pipeline

See Tensorflow's ▸ Better performance with the tf.data API

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

IT4Innovations National Supercomputing Center

VŠB – Technical University of Ostrava
Studentská 6231/1B
708 00 Ostrava-Poruba, Czech Republic
www.it4i.cz

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research, Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS