Discussion 6
Introduction to Tensorflow.Keras

EE599 Deep Learning

Kuan-Wen (James) Huang

Spring 2020

Installation

- Activate your virtual environment, either Anaconda or pyenv
- Install Tensorflow via
  - pip install tensorflow (works for both Anaconda or pyenv)
  - Install tensorflow package with Anaconda-navigator
- For GPU support (available for Ubuntu and Windows with CUDA®-enabled cards), check https://www.tensorflow.org/install/gpu

Guides and References of tf.Keras

- Guide:
  https://www.tensorflow.org/guide/keras/overview

- Tutorial:
  https://www.tensorflow.org/tutorials

- Reference:
  https://www.tensorflow.org/api_docs/python/tf/keras/

Two ways of using tf.Keras

- Sequential model
  - Easy
  - Less freedom

- Functional API model
  - Good for complex models
    - Models with branches, multiple inputs or multiple outputs
    - Layers with shared parameters
    - Custom layer, output
    - Custom functions

Sequential Model – example 1

- Check 1_fmnist.py
- Create a model with keras.Sequential() with a list of layer objects including input, hidden layers and output, e.g.

```
model = keras.Sequential([
    keras.layers.Input(shape=(784,),name='input'),
    keras.layers.Dense(128, activation='relu', name='hidden1'),
    keras.layers.Dense(10, activation='softmax', name='output')
])
```

Sequential Model – example 2

```
model = keras.Sequential()
model.add(keras.Layers.Dense(128, input_shape=(784,)))

# now the model will take as input arrays of shape (*, 784)
# and output arrays of shape (*, 128)
# after the first layer, you don't need to specify
# the size of the input anymore

model.add(keras.Layers.Activation('relu'))
# add an activation layer

model.add(keras.Layers.Dense(10))
```

Sequential Model – Layer objects

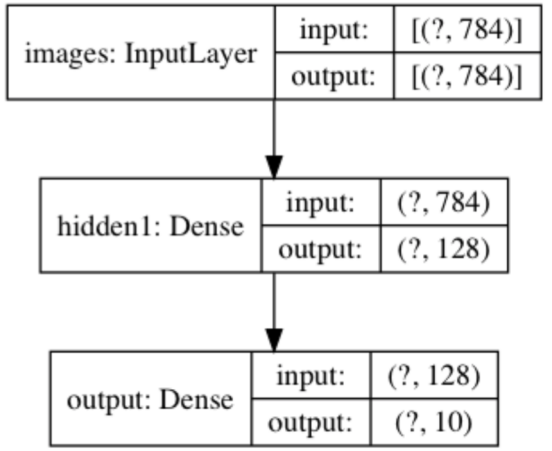
- For dense layer, check https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

- Arguments:

```
__init__(
    units,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

Print summary of model

- `model.summary()`



- To produce a digram of the model
- `keras.utils.plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=True)`

- This requires pydot and graphviz installed (pip install pydot graphviz)
- You may need to update graphviz package first (Mac: brew install graphviz/Linux: sudo apt-get install graphviz)

Compile the model

```
compile(
    optimizer='rmsprop',
    loss=None,
    metrics=None,
    loss_weights=None,
    sample_weight_mode=None,
    weighted_metrics=None,
    target_tensors=None,
    distribute=None,
    **kwargs
)
```

- Use the method compile() of keras.Model
  - `model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy’])`
- You may set your own optimizer and loss:
  - `opt_sgd = keras.optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9)`
  - `loss_ssc = keras.losses.SparseCategoricalCrossentropy()`
  - `model.compile(loss=loss_ssc, optimizer=opt_sgd', metrics=[’accuracy’])`
- https://www.tensorflow.org/api_docs/python/tf/keras/optimizers
- https://www.tensorflow.org/api_docs/python/tf/keras/losses

Fit the training data

- `results = model.fit(train_images,  train_labels, batch_size=32, epochs=10, validation_split=0.1)`

- fit() method of keras.Model returns a History object
- The attribute of History.history is a dictionary of training loss, validation loss & training metrics and validation matrics recorded at every epoch

- You can see the keys via `results.history.keys()`
- https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit

Same story with functional API

- Check 2_fmnist.py

```
in = keras.layers.Input(shape=(784,), name='input')
h0 = keras.layers.Dense(128, activation='relu', name='hidden0')(in)
h1 = keras.layers.Dense(64, activation='relu', name='hidden1')(h0)
out = keras.layers.Dense(10, activation='softmax', name='output')(h1)

hidden_layers = [h0, h1]
model = keras.Model(inputs=in, outputs=out)
```

- Then you can compile the model with specified optimizer, loss and metrics and fit the model with training data.

Functional API with branches

- Check disccusion6_mnist_branches.ipynb
- Functional API can handle models with branches, multiple inputs and outputs
- Use keras.layers.concatenate([list of layers]) to join branches into a layer

Load, delete and save models

- `model.save('fmnist_trained.hdf5') # saves the model`
- `del model # deletes the model`
- `model = keras.models.load_model('fmnist_trained.hdf5')`
  `# loads the model`
- Save and load model weights
- `model.save_weights('my_model_weights.hdf5')`
  `model.load_weights('my_model_weights.hdf5')`

Freeze a layer for fine-tuning

- When you want to update only some layers for new data (fine-tuning the model):
- Pass the trainable argument in the layer object
  - `frozen_layer = keras.Layers.Dense(30, trainable=False)`
- Or set the trainable attribute to False
  - `# hidden_layers[0] is a layer`
  - `hidden_layers[0].trainable = False`

More on keras.Model.fit()

- **verbose**: 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch.

- **validation_split**: Float between 0 and 1. Fraction of the training data to be used as validation data.

- **shuffle**: Boolean (whether to shuffle the training data before each epoch)

```
fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    verbose=1,
    callbacks=None,
    validation_split=0.0,
    validation_data=None,
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False,
    **kwargs
)
```

More on keras.Model.fit()

- **callbacks**:
  List of keras.callbacks.Callback instances.
  - A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training.

```
fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    verbose=1,
    callbacks=None,
    validation_split=0.0,
    validation_data=None,
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False,
    **kwargs
)
```

callbacks

- Gives us ability to save model at certain stage of training, log loss and other statistics of the model, etc.

- You may use classes provided by keras.callbacks
- `mcp = keras.callbacks.ModelCheckpoint(filepath='/saved/weights.hdf5', verbose=1, save_best_only=True)`
- `model.fit(train_images, train_labels, batch_size=32, epochs=25, validation_split=0.1, callbacks=[mcp])`

- Or write your own callbacks which inherit the methods but with custom stuff.
- See tf.keras.callbacks https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/Callback