Discussion 7
More on Tensorflow.Keras

EE599 Deep Learning

Kuan-Wen (James) Huang

Spring 2020

Contributors: Kuan-Wen Huang , Arindam Jati

Last time

1. Sequential vs. function API
2. After you build the model with keras.Layers objects, you have to .compile() it with optimizer and loss function. Common optimizers: SGD, Adam, RMSprop
3. Loss functions: SparseCategoricalCrossentropy handles true labels as integers while CategoricalCrossentropy expect true label to be one-hot vectors.
4. After you compile your model, you use .fit() to train the net with X_train, y_train and selected batch_size and number of epochs to train. You can specify the validation split as well.

Keras 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='models/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])
```

- See discussion7_callbacks.ipynb
- https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/Callback

Custom callbacks

- What if things given in the module `keras.callbacks` do not meet our requirement?
  - E.g. we want to save results including model, loss, accuracy at some given stages of training procedure.

- We want to customize a set of function to be applied during training.

- We can do it by inheriting `keras.callbacks.Callback` and overloading its methods, e.g. `on_batch_begin, on_batch_end, on_epoch_begin, on_epoch_end`, etc.

Custom callbacks

```
# Define Custom Callback
class AccuracyLogger(keras.callbacks.Callback): # Inherit from base class
    def on_train_begin(self, logs={}):          # Overload method
        print('Creating a log directory')       # Do sth at the beginning of
        os.system('mkdir -p logs')              # the entire training procedure
        with open('logs/train_acc.txt', 'w') as f:
            f.write('')
        with open('logs/val_acc.txt', 'w') as f:
            f.write('')
    def on_epoch_end(self, epoch, logs={}):     # Overload method
        print('Inside callback: epoch = ', epoch, 'logs = ', logs)
        train_acc = logs['accuracy']            # Do sth at the end of epoch
        val_acc = logs['val_accuracy']
        with open('logs/train_acc.txt', 'a') as f: # open in append mode 'a'
            f.write('Epoch {}: {}\n'.format(epoch, train_acc))
        with open('logs/val_acc.txt', 'a') as f: # open in append mode 'a'
            f.write('Epoch {}: {}\n'.format(epoch, val_acc))
```

Custom callbacks

- See discussion7_custom_callbacks.ipynb

Custom Layer

- There are already tons of layer classes you can use, including `keras.layers.Dense`, `keras.layers.Dropuout`, `keras.layers.Concatenate`, `keras.layers.Conv2D`, `keras.layers.LeakyReLU`, etc.

- In case you are not satisfied, you can write your own custom layer by inheriting `keras.layers.Layer`

- Two usual methods to overload: `build()` and `call()`.
- `build()` is called once after it knows the input shape and dtype.
- `call()` is called when applying the layer to input tensors.

Custom Layer

```
class WeirdLinear(keras.layers.Layer):

  def __init__(self, units=10):
    super(WeirdLinear, self).__init__() # same as keras.layers.Layer.__init__()
    self. units= units

  def build(self, input_shape):
    self.w = self.add_weight(shape=(input_shape[-1], self.units),
                             name = 'w', initializer='random_normal',
                             trainable=False)
    self.b = self.add_weight(shape=(self.units,),
                             name = 'b',initializer='random_normal',
                             trainable=True)

  def call(self, inputs):
    return inputs @ self.w + self.b   # shape of self.b broadcasting


# create an instance of WeirdLinear as the first hidden layer

wl = WeirdLinear(100)(inputs)
```

Custom Layer

- See discussion7_custom_layer_loss_metric.ipynb

- A good practice when writing custom layer is to check https://www.tensorflow.org/api_docs/python/tf/keras/layers/Layer first to see whether there are similar layer class defined.

Custom Loss and Custom Metric

- Utilize `tensorflow.keras.backend` module, it helps us handle low-level math operations (convolution, correlation, product, exponential, etc on Tensor objects) and return a Tensor object.

```
from tensorflow.keras import backend as K

# Custom Loss
class WeightedCCE(keras.losses.Loss): # Inherit from keras.losses.Loss
    def __init__(self, weights):
        super(WeightedCCE, self).__init__()
        self.weights = weights

    def call(self, y_true, y_pred):
        return - K.mean(K.sum(y_true * K.log(y_pred) * self.weights,axis = 1))
```

Custom Loss and Custom Metric

```
# Custom Metric to log
def max_y_predict(y_true, y_pred):
    return K.max(y_pred)

# Custom Loss Object
wcce3 = WeightedCCE([20, 20, 1, 1, 1, 2, 3, 4, 5, 6])

model.compile(optimizer='sgd',
              loss=wcce3, # Custom loss object
              metrics=['accuracy', max_y_predict])  # Custom metric
```

- See discussion7_custom_layer_loss_metric.ipynb