‹ Return to "Artificial Intelligence Nanodegree and Specializations" in the classroom

DISCUSS ON STUDENT HUB

DNN Speech Recognizer

REVIEW

CODE REVIEW 5

HISTORY

▾ sample_models.py 5

```python
from keras import backend as K
from keras.models import Model
from keras.layers import (BatchNormalization, Conv1D, Dense, Input,
    TimeDistributed, Activation, Bidirectional, SimpleRNN, GRU, LSTM)

def simple_rnn_model(input_dim, output_dim=29):
    """ Build a recurrent network for speech
    """
    # Main acoustic input
    input_data = Input(name='the_input', shape=(None, input_dim))
    # Add recurrent layer
    simp_rnn = GRU(output_dim, return_sequences=True,
                 implementation=2, name='rnn')(input_data)
    # Add softmax activation layer
    y_pred = Activation('softmax', name='softmax')(simp_rnn)
    # Specify the model
    model = Model(inputs=input_data, outputs=y_pred)
    model.output_length = lambda x: x
    print(model.summary())
    return model

def rnn_model(input_dim, units, activation, output_dim=29):
    """ Build a recurrent network for speech
    """
    # Main acoustic input
    input_data = Input(name='the_input', shape=(None, input_dim))
    # Add recurrent layer
    simp_rnn = GRU(units, activation=activation,
        return_sequences=True, implementation=2, name='rnn')(input_data)
```

SUGGESTION

You can experiment with different architectures here. For example LSTM, SimpleRNN.

```
    # TODO: Add batch normalization
    bn_rnn = BatchNormalization()(simp_rnn)
    # TODO: Add a TimeDistributed(Dense(output_dim)) layer
    time_dense = TimeDistributed(Dense(output_dim))(bn_rnn)
    # Add softmax activation layer
    y_pred = Activation('softmax', name='softmax')(time_dense)
    # Specify the model
    model = Model(inputs=input_data, outputs=y_pred)
    model.output_length = lambda x: x
    print(model.summary())
```

AWESOME

As you can see from the results, model 1 is a significant improvement over model 0. It incorporates GI capture the variance of the data, and the TimeDistributed layer also allows the model to convey the tr significantly :)

```
    return model


def cnn_rnn_model(input_dim, filters, kernel_size, conv_stride,
    conv_border_mode, units, output_dim=29):
    """ Build a recurrent + convolutional network for speech
    """
    # Main acoustic input
    input_data = Input(name='the_input', shape=(None, input_dim))
    # Add convolutional layer
    conv_1d = Conv1D(filters, kernel_size,
                     strides=conv_stride,
                     padding=conv_border_mode,
                     activation='relu',
                     name='conv1d')(input_data)
    # Add batch normalization
    bn_cnn = BatchNormalization(name='bn_conv_1d')(conv_1d)
    # Add a recurrent layer
    simp_rnn = SimpleRNN(units, activation='relu',
```

SUGGESTION

You can use dropouts here to avoid overfitting.

```
        return_sequences=True, implementation=2, name='rnn')(bn_cnn)
    # TODO: Add batch normalization
    bn_rnn = BatchNormalization(name='bn_rnn')(simp_rnn)
    # TODO: Add a TimeDistributed(Dense(output_dim)) layer
    time_dense = TimeDistributed(Dense(output_dim))(bn_rnn)
    # Add softmax activation layer
    y_pred = Activation('softmax', name='softmax')(time_dense)
    # Specify the model
    model = Model(inputs=input_data, outputs=y_pred)
    model.output_length = lambda x: cnn_output_length(
        x, kernel_size, conv_border_mode, conv_stride)
    print(model.summary())
```

AWESOME

model 2 further improves the performance over model 1 by adding a 1D convolutional layer, which all
architecture captures relationships that account for the context.

```
    return model

def cnn_output_length(input_length, filter_size, border_mode, stride,
                       dilation=1):
    """ Compute the length of the output sequence after 1D convolution alor
        time. Note that this function is in line with the function used in
        Convolution1D class from Keras.
    Params:
        input_length (int): Length of the input sequence.
        filter_size (int): Width of the convolution kernel.
        border_mode (str): Only support `same` or `valid`.
        stride (int): Stride size used in 1D convolution.
        dilation (int)
    """
    if input_length is None:
        return None
    assert border_mode in {'same', 'valid'}
    dilated_filter_size = filter_size + (filter_size - 1) * (dilation - 1)
    if border_mode == 'same':
        output_length = input_length
    elif border_mode == 'valid':
        output_length = input_length - dilated_filter_size + 1
    return (output_length + stride - 1) // stride

def deep_rnn_model(input_dim, units, recur_layers, output_dim=29):
    """ Build a deep recurrent network for speech
    """
    # Main acoustic input
    input_data = Input(name='the_input', shape=(None, input_dim))
    # TODO: Add recurrent layers, each with batch normalization
    if recur_layers == 1:
        layer = GRU(units, activation='relu', return_sequences=True,
                    implementation=2, name='rnn_1')(input_data)
        layer = BatchNormalization(name='bn_rnn_1')(layer)
    else:
        layer = GRU(units, activation='relu', return_sequences=True,
                    implementation=2, name='rnn_1')(input_data)
        layer = BatchNormalization(name='bn_rnn_1')(layer)

        for i in range(recur_layers - 2):
            layer = GRU(units, activation='relu', return_sequences=True,
                        implementation=2, name='rnn_{}'.format(2+i))(layer)
            layer = BatchNormalization(name='bn_rnn_{}'.format(2+i))(layer)

        layer = GRU(units, activation='relu', return_sequences=True,
                    implementation=2, name='rnn_last')(layer)
        layer = BatchNormalization(name='bt_rnn_last')(layer)
    # TODO: Add a TimeDistributed(Dense(output_dim)) layer
    time_dense = TimeDistributed(Dense(output_dim))(layer)
    # Add softmax activation layer
    y_pred = Activation('softmax', name='softmax')(time_dense)
    # Specify the model
    model = Model(inputs=input_data, outputs=y_pred)
    model.output_length = lambda x: x
    print(model.summary())
    return model

def bidirectional_rnn_model(input_dim, units, output_dim=29):
```

```
    """ Build a bidirectional recurrent network for speech
    """
    # Main acoustic input
    input_data = Input(name='the_input', shape=(None, input_dim))
    # TODO: Add bidirectional recurrent layer
    bd_rnn = Bidirectional(GRU(units, activation="relu",return_sequences=Tr
                 implementation=2, name="bd_rnn"))(input_data)
```

AWESOME

Nicely implemented. To add dropout to recurrent layers, pay special attention to the dropout_W and c

```
    # TODO: Add a TimeDistributed(Dense(output_dim)) layer
    time_dense = TimeDistributed(Dense(output_dim))(bd_rnn)
    # Add softmax activation layer
    y_pred = Activation('softmax', name='softmax')(time_dense)
    # Specify the model
    model = Model(inputs=input_data, outputs=y_pred)
    model.output_length = lambda x: x
    print(model.summary())
    return model

def final_model(input_dim, filters, kernel_size, conv_stride, conv_border_m
                units, output_dim=29, dropout_rate=0.5, number_of_layers=2,
                cell=GRU, activation='relu'):
    """ Build a deep network for speech
    """
    # Main acoustic input
    input_data = Input(name='the_input', shape=(None, input_dim))
    # TODO: Specify the layers in your network
    conv_1d = Conv1D(filters, kernel_size,
                     strides=conv_stride,
                     padding=conv_border_mode,
                     activation='relu',
                     name='layer_1_conv',
                     dilation_rate=1)(input_data)
    conv_bn = BatchNormalization(name='conv_batch_norm')(conv_1d)


    if number_of_layers == 1:
        layer = cell(units, activation=activation,return_sequences=True,
                     implementation=2, name='rnn_1', dropout=dropout_rate)(
        layer = BatchNormalization(name='bn_rnn_1')(layer)
    else:
        layer = cell(units, activation=activation,return_sequences=True,
                     implementation=2, name='rnn_1', dropout=dropout_rate)(
        layer = BatchNormalization(name='bn_rnn_1')(layer)

        for i in range(number_of_layers - 2):
            layer = cell(units, activation=activation,return_sequences=True
                         implementation=2, name='rnn_{}'.format(i+2),
                         dropout=dropout_rate)(layer)
            layer = BatchNormalization(name='bn_rnn_{}'.format(i+2))(layer)

        layer = cell(units, activation=activation,return_sequences=True,
                     implementation=2, name='last_rnn')(layer)
        layer = BatchNormalization(name='bn_rnn_last')(layer)

    time_dense = TimeDistributed(Dense(output_dim))(layer)
    # TODO: Add softmax activation layer
    y_pred = Activation('softmax', name='softmax')(time_dense)
    # Specify the model
    model = Model(inputs=input_data, outputs=y_pred)
```

```
    # TODO: Specify model.output_length
    model.output_length = lambda x: cnn_output_length(x, kernel_size,
                                                      conv_border_mode,
                                                      conv_stride)
    print(model.summary())
    return model
```

▸ workspace_utils.py

▸ utils.py

▸ train_utils.py

▸ data_generator.py

▸ char_map.py

RETURN TO PATH

Rate this review