

[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

```


1 from keras import backend as K
2 from keras.models import Model
3 from keras.layers import (BatchNormalization, Conv1D, Dense, Input,
4     TimeDistributed, Activation, Bidirectional, SimpleRNN, GRU, LSTM)
5
6 def simple_rnn_model(input_dim, output_dim=29):
7     """ Build a recurrent network for speech
8     """
9     # Main acoustic input
10    input_data = Input(name='the_input', shape=(None, input_dim))
11    # Add recurrent layer
12    simp_rnn = GRU(output_dim, return_sequences=True,
13        implementation=2, name='rnn')(input_data)
14    # Add softmax activation layer
15    y_pred = Activation('softmax', name='softmax')(simp_rnn)
16    # Specify the model
17    model = Model(inputs=input_data, outputs=y_pred)
18    model.output_length = lambda x: x
19    print(model.summary())
20    return model
21
22 def rnn_model(input_dim, units, activation, output_dim=29):
23     """ Build a recurrent network for speech
24     """
25     # Main acoustic input
26    input_data = Input(name='the_input', shape=(None, input_dim))
27    # Add recurrent layer
28    simp_rnn = GRU(units, activation=activation,
29        return_sequences=True, implementation=2, name='rnn')(input_data)

```

## SUGGESTION

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


```
30 # TODO: Add batch normalization
31 bn_rnn = BatchNormalization()(simp_rnn)
32 # TODO: Add a TimeDistributed(Dense(output_dim)) layer
33 time_dense = TimeDistributed(Dense(output_dim))(bn_rnn)
34 # Add softmax activation layer
35 y_pred = Activation('softmax', name='softmax')(time_dense)
36 # Specify the model
37 model = Model(inputs=input_data, outputs=y_pred)
38 model.output_length = lambda x: x
39 print(model.summary())
```



## AWESOME

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

```
40 return model
41
42
43 def cnn_rnn_model(input_dim, filters, kernel_size, conv_stride,
44                  conv_border_mode, units, output_dim=29):
45     """ Build a recurrent + convolutional network for speech """
46
47     # Main acoustic input
48     input_data = Input(name='the_input', shape=(None, input_dim))
49     # Add convolutional layer
50     conv_1d = Conv1D(filters, kernel_size,
51                     strides=conv_stride,
52                     padding=conv_border_mode,
53                     activation='relu',
54                     name='conv1d')(input_data)
55     # Add batch normalization
56     bn_cnn = BatchNormalization(name='bn_conv_1d')(conv_1d)
57     # Add a recurrent layer
58     simp_rnn = SimpleRNN(units, activation='relu',
```



## SUGGESTION

You can use dropouts here to avoid overfitting.

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

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.

```

71     return model
72
73 def cnn_output_length(input_length, filter_size, border_mode, stride,
74                       dilation=1):
75     """ Compute the length of the output sequence after 1D convolution alor
76         time. Note that this function is in line with the function used in
77         Convolution1D class from Keras.
78     Params:
79         input_length (int): Length of the input sequence.
80         filter_size (int): Width of the convolution kernel.
81         border_mode (str): Only support `same` or `valid`.
82         stride (int): Stride size used in 1D convolution.
83         dilation (int)
84     """
85     if input_length is None:
86         return None
87     assert border_mode in {'same', 'valid'}
88     dilated_filter_size = filter_size + (filter_size - 1) * (dilation - 1)
89     if border_mode == 'same':
90         output_length = input_length
91     elif border_mode == 'valid':
92         output_length = input_length - dilated_filter_size + 1
93     return (output_length + stride - 1) // stride
94
95 def deep_rnn_model(input_dim, units, recur_layers, output_dim=29):
96     """ Build a deep recurrent network for speech
97     """
98     # Main acoustic input
99     input_data = Input(name='the_input', shape=(None, input_dim))
100    # TODO: Add recurrent layers, each with batch normalization
101    if recur_layers == 1:
102        layer = GRU(units, activation='relu', return_sequences=True,
103                    implementation=2, name='rnn_1')(input_data)
104        layer = BatchNormalization(name='bn_rnn_1')(layer)
105    else:
106        layer = GRU(units, activation='relu', return_sequences=True,
107                    implementation=2, name='rnn_1')(input_data)
108        layer = BatchNormalization(name='bn_rnn_1')(layer)
109
110        for i in range(recur_layers - 2):
111            layer = GRU(units, activation='relu', return_sequences=True,
112                        implementation=2, name='rnn_{}'.format(2+i))(layer)
113            layer = BatchNormalization(name='bn_rnn_{}'.format(2+i))(layer)
114
115        layer = GRU(units, activation='relu', return_sequences=True,
116                    implementation=2, name='rnn_last')(layer)
117        layer = BatchNormalization(name='bt_rnn_last')(layer)
118    # TODO: Add a TimeDistributed(Dense(output_dim)) layer
119    time_dense = TimeDistributed(Dense(output_dim))(layer)
120    # Add softmax activation layer
121    y_pred = Activation('softmax', name='softmax')(time_dense)
122    # Specify the model
123    model = Model(inputs=input_data, outputs=y_pred)
124    model.output_length = lambda x: x
125    print(model.summary())
126    return model
127
128 def bidirectional_rnn_model(input_dim, units, output_dim=29):

```

```

129     """ Build a bidirectional recurrent network for speech
130     """
131     # Main acoustic input
132     input_data = Input(name='the_input', shape=(None, input_dim))
133     # TODO: Add bidirectional recurrent layer
134     bd_rnn = Bidirectional(GRU(units, activation="relu", return_sequences=True,
135                               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

```

136     # TODO: Add a TimeDistributed(Dense(output_dim)) layer
137     time_dense = TimeDistributed(Dense(output_dim))(bd_rnn)
138     # Add softmax activation layer
139     y_pred = Activation('softmax', name='softmax')(time_dense)
140     # Specify the model
141     model = Model(inputs=input_data, outputs=y_pred)
142     model.output_length = lambda x: x
143     print(model.summary())
144     return model
145
146 def final_model(input_dim, filters, kernel_size, conv_stride, conv_border_mode,
147                 units, output_dim=29, dropout_rate=0.5, number_of_layers=2,
148                 cell=GRU, activation='relu'):
149     """ Build a deep network for speech
150     """
151     # Main acoustic input
152     input_data = Input(name='the_input', shape=(None, input_dim))
153     # TODO: Specify the layers in your network
154     conv_1d = Conv1D(filters, kernel_size,
155                      strides=conv_stride,
156                      padding=conv_border_mode,
157                      activation='relu',
158                      name='layer_1_conv',
159                      dilation_rate=1)(input_data)
160     conv_bn = BatchNormalization(name='conv_batch_norm')(conv_1d)
161
162     if number_of_layers == 1:
163         layer = cell(units, activation=activation, return_sequences=True,
164                      implementation=2, name='rnn_1', dropout=dropout_rate)(
165             layer = BatchNormalization(name='bn_rnn_1')(layer)
166     else:
167         layer = cell(units, activation=activation, return_sequences=True,
168                      implementation=2, name='rnn_1', dropout=dropout_rate)(
169             layer = BatchNormalization(name='bn_rnn_1')(layer)
170
171         for i in range(number_of_layers - 2):
172             layer = cell(units, activation=activation, return_sequences=True,
173                          implementation=2, name='rnn_{}'.format(i+2),
174                          dropout=dropout_rate)(layer)
175             layer = BatchNormalization(name='bn_rnn_{}'.format(i+2))(layer)
176
177         layer = cell(units, activation=activation, return_sequences=True,
178                      implementation=2, name='last_rnn')(layer)
179         layer = BatchNormalization(name='bn_rnn_last')(layer)
180
181     time_dense = TimeDistributed(Dense(output_dim))(layer)
182     # TODO: Add softmax activation layer
183     y_pred = Activation('softmax', name='softmax')(time_dense)
184     # Specify the model
185     model = Model(inputs=input_data, outputs=y_pred)

```

```
187 # TODO: Specify model.output_length
188 model.output_length = lambda x: cnn_output_length(x, kernel_size,
189                                                    conv_border_mode,
190                                                    conv_stride)
191 print(model.summary())
192 return model
```

► workspace\_utils.py

► utils.py

► train\_utils.py

► data\_generator.py

► char\_map.py

RETURN TO PATH

Rate this review

---