

[◀ Return to Classroom](#)[DISCUSS ON STUDENT HUB](#)

DNN Speech Recognizer

[REVIEW](#)[CODE REVIEW 6](#)[HISTORY](#)[▼ sample_models.py 6](#)

```
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, MaxPool1D, Dropout)
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)
30     # TODO: Add batch normalization
31     bn_rnn = BatchNormalization(name='bn_rnn')(simp_rnn)

```

SUGGESTION

Best use of BatchNorm can be done just after input layer

```

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())
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

Try using a better RNN network like GRU here

```

59     return_sequences=True, implementation=2, name='rnn')(bn_cnn)
60 # TODO: Add batch normalization
61 bn_rnn = BatchNormalization(name='bn_simp_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())
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 along
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)

```

```

88     if border_mode == 'same':
89         output_length = input_length
90     elif border_mode == 'valid':
91         output_length = input_length - dilated_filter_size + 1
92     return (output_length + stride - 1) // stride
93
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     for i in range(recur_layers):

```



AWESOME

Well done on robust implementation using `recur_layers`

```

102         if i==0:
103             simp_rnn = GRU(units, return_sequences=True)(input_data)
104         else:
105             simp_rnn = GRU(units, return_sequences=True)(bn_rnn)
106             bn_rnn = BatchNormalization()(simp_rnn)
107         # TODO: Add a TimeDistributed(Dense(output_dim)) layer
108         time_dense = TimeDistributed(Dense(output_dim))(bn_rnn)
109         # Add softmax activation layer
110         y_pred = Activation('softmax', name='softmax')(time_dense)
111         # Specify the model
112         model = Model(inputs=input_data, outputs=y_pred)
113         model.output_length = lambda x: x
114         print(model.summary())
115         return model
116
117
118 def bidirectional_rnn_model(input_dim, units, output_dim=29):
119     """ Build a bidirectional recurrent network for speech
120     """
121     # Main acoustic input
122     input_data = Input(name='the_input', shape=(None, input_dim))
123     # TODO: Add bidirectional recurrent layer
124     bidir_rnn = Bidirectional(GRU(units, return_sequences=True, name = 'bidir_rnn'), merge_mode='concat')(input_data)

```



Correct Implementation of Bidirectional layer

```

125 # TODO: Add a TimeDistributed(Dense(output_dim)) layer
126 time_dense = TimeDistributed(Dense(output_dim))(bidir_rnn)
127 # Add softmax activation layer
128 y_pred = Activation('softmax', name='softmax')(time_dense)
129 # Specify the model
130 model = Model(inputs=input_data, outputs=y_pred)
131 model.output_length = lambda x: x
132 print(model.summary())
133 return model
134
135
136 # def final_model(input_dim, filters, kernel_size, conv_stride,
137 # conv_border_mode, units, output_dim=29):
138 # """ Build a deep network for speech
139 # """
140 # # Main acoustic input
141 # input_data = Input(name='the_input', shape=(None, input_dim))
142 # # TODO: Specify the layers in your network
143
144 # # Add convolutional layer
145 # conv_1d = Conv1D(filters, kernel_size,
146 #                 strides=conv_stride,
147 #                 padding=conv_border_mode,
148 #                 activation='relu',
149 #                 name='conv1d')(input_data)
150 # # dropout layer
151 # drop = Dropout(0.2)(conv_1d)
152 # # Add batch normalization
153 # bn_cnn = BatchNormalization(name='bn_conv_1d')(drop)
154
155 # Add bidirectional recurrent layer
156 # bidir_rnn = Bidirectional(GRU(units, return_sequences=True, name = 'bidir_rnn'), merge_mode='concat')(bn_cnn)
157 # bn_bi_rnn = BatchNormalization(name='bn__bi_rnn')(bidir_rnn)
158
159 # bidir_rnn = Bidirectional(GRU(units, activation='relu',
160 # return_sequences=True, implementation=2, name='bid_rnn0', dropout=0.2))(bn_cnn)
161 # bn_bi_rnn = BatchNormalization(name='bn_rnn1')(bidir_rnn)
162 # bidir_rnn = Bidirectional(GRU(units, activation='relu',
163 # return_sequences=True, implementation=2, name='bid_rnn1', dropout=0.2))(bn_bi_rnn)
164 # bn_bi_rnn = BatchNormalization(name='bn_rnn2')(bidir_rnn)

```

```

165
167 # # Add two RNN layers
168 # simp_rnn1 = GRU(units, return_sequences=True, dropout_W=0.0, dropout_U=0.1)(bn_cnn)
169 # bn_rnn1 = BatchNormalization(name='bn_conv_1d1')(simp_rnn1)
170 # simp_rnn2 = GRU(units, return_sequences=True, dropout_W=0.5, dropout_U=0.1)(bn_rnn1)
171 # bn_rnn2 = BatchNormalization(name='bn_conv_1d2')(simp_rnn2)
172
173 # # Add three bidirectional recurrent layers
174 # bidir_rnn = Bidirectional(GRU(units, return_sequences=True, implementation=2, name='bidirnn0', dropout=0.2, recurren
175 # bidir_rnn = Bidirectional(GRU(units, return_sequences=True, implementation=2, name='bidirnn1', dropout=0.2, recurren
176 # bidir_rnn = Bidirectional(GRU(units, return_sequences=True, implementation=2, name='bidirnn2', dropout=0.2, recurren
177
178
179 # Add a TimeDistributed(Dense(output_dim)) layer
180 # time_dense = TimeDistributed(Dense(output_dim))(bn_rnn2)
181 # # TODO: Add softmax activation layer
182 # y_pred = Activation('softmax', name='softmax')(time_dense)
183
184 # # Specify the model
185 # model = Model(inputs=input_data, outputs=y_pred)
186
187 # # Specify model.output_length
188 # model.output_length = lambda x: cnn_output_length(
189 #     x, kernel_size, conv_border_mode, conv_stride)
190
191 # print(model.summary())
192 # return model
193
194 def final_model(input_dim, filters, kernel_size, conv_stride,
195                 conv_border_mode, units, output_dim=29):
196     """ Build a deep network for speech
197         cnn + 2 bidirectional recurrent network for speech
198     """
199     # Main acoustic input
200     input_data = Input(name='the_input', shape=(None, input_dim))
201     # TODO: Specify the layers in your network
202     # Add convolutional layer
203     conv_1d = Conv1D(filters, kernel_size,
204                      strides=conv_stride,
205                      padding=conv_border_mode,
206                      activation='relu',
207                      name='conv1d')(input_data)
208
209 # # dropout layer
210 # conv_1d = Dropout(0.2)(conv_1d)
211 # Add batch normalization

```

```

212     bn_cnn = BatchNormalization(name='bn_conv_1d')(conv_1d)
213
214     # Add two RNN layers
215     simp_rnn1 = GRU(units, return_sequences=True, dropout_W=0.0, dropout_U=0.1)(bn_cnn)

```

SUGGESTION

Try using the concept of `recur_layers` here

You may remove `dropout_W` if you are setting it to 0

```

216     bn_rnn1 = BatchNormalization(name='bn_conv_1d1')(simp_rnn1)
217     simp_rnn2 = GRU(units, return_sequences=True, dropout_W=0.5, dropout_U=0.1)(bn_rnn1)

```

AWESOME

Well done on using dropout here.

You may also add dropout layer

```

218     bn_rnn2 = BatchNormalization(name='bn_conv_1d2')(simp_rnn2)
219
220     # Add a TimeDistributed(Dense(output_dim)) layer
221     time_dense = TimeDistributed(Dense(output_dim))(bn_rnn2)
222
223     # TODO: Add softmax activation layer
224     y_pred = Activation('softmax', name='softmax')(time_dense)
225     # Specify the model
226     model = Model(inputs=input_data, outputs=y_pred)
227     # TODO: Specify model.output_length
228     model.output_length = lambda x: cnn_output_length(
229         x, kernel_size, conv_border_mode, conv_stride)
230     print(model.summary())
231     return model
232
233
234
235

```

▶ workspace-utils.py

▶ utils.py

▶ train_utils.py

▶ data_generator.py

▶ char_map.py

RETURN TO PATH
