Estimate generation v2

Use advanced models for generation estimation in the Global Power Plant Database. Primary model is a two-hidden-layer neural network.

In [1]:

```
# import what we'll need and set parameters

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Flatten, Dense, Lambda
from keras.layers import Conv2D, Dropout, Activation, MaxPooling2D
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.utils.vis_utils import model_to_dot
from IPython.display import SVG
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import pydot

GPPD_FILENAME = '../../output_database/global_power_plant_database.csv'
WEIGHTS_FILE = "model/estimate_generation.h5"
VALIDATION_FRACTION = 0.2
```

```
Using TensorFlow backend.
```

In [2]:

```
# read in database
df = pd.read_csv(GPPD_FILENAME)
df.head()
```

Out[2]:

	country	country_long	name	gppd_idnr	capacity_mw	latitude	longitu
0	AFG	Afghanistan	Kajaki Hydroelectric Power Plant Afghanistan	GEODB0040538	33.00	32.3220	65.119
1	AFG	Afghanistan	Mahipar Hydroelectric Power Plant Afghanistan	GEODB0040541	66.00	34.5560	69.478
2	AFG	Afghanistan	Naghlu Dam Hydroelectric Power Plant Afghanistan	GEODB0040534	100.00	34.6410	69.717
3	AFG	Afghanistan	Nangarhar (Darunta) Hydroelectric Power Plant ...	GEODB0040536	11.55	34.4847	70.363
4	AFG	Afghanistan	Northwest Kabul Power Plant Afghanistan	GEODB0040540	42.00	34.5638	69.113

5 rows × 22 columns

In [3]:

```
# show count for number of valid entries in each column
df.count()
```

Out[3]:

```
country                    25657
country_long               25657
name                       25637
gppd_idnr                  25657
capacity_mw                25657
latitude                   25657
longitude                  25657
fuel1                      25657
fuel2                       1670
fuel3                        295
fuel4                        107
commissioning_year         13933
owner                      17157
source                     25657
url                        25657
geolocation_source         25657
year_of_capacity_data      16065
generation_gwh_2013           56
generation_gwh_2014           55
generation_gwh_2015          536
generation_gwh_2016         8657
estimated_generation_gwh   24941
dtype: int64
```

In [4]:

```
# prepare data for training

# convert string-type columns to categories (assume no NaNs in these columns)
factorized_countries,country_key = df['country'].factorize()
df['country'] = factorized_countries
factorized_fuel1,fuel1_key = df['fuel1'].factorize()
df['fuel1'] = factorized_fuel1

# convert numerical columns to np array to use as predictor variables and remove
NaNs
X_columns = ['country','capacity_mw','latitude','longitude','commissioning_year'
,'fuel1']
df_No_NaN = df[X_columns + ['generation_gwh_2016']].dropna(how='any')
X_data = df_No_NaN[X_columns].as_matrix()

# calculate capacity factor to use as predicted variable
df_No_NaN['capacity_factor'] = df_No_NaN.apply(lambda row:row['generation_gwh_20
16']/(24.0*365.0*0.001*row['capacity_mw']),axis=1)
y_column = ['capacity_factor']
y_data = df_No_NaN[y_column].as_matrix()

# show results
print(X_data)
print(y_data)
print(len(X_data))
print(len(y_data))
```

```
[[  8.00000000e+00   2.89000000e+02   4.72078000e+01   1.10057000e+0
1
    1.98100000e+03   0.00000000e+00]
 [  8.00000000e+00   5.00000000e+02   4.72696000e+01   1.09678000e+0
1
    1.98100000e+03   0.00000000e+00]
 [  4.30000000e+01   2.25000000e+03   3.02483000e+01   3.09471000e+0
1
    2.01400000e+03   1.00000000e+00]
 ...,
 [  1.57000000e+02   2.80000000e+01   1.43611000e+01   1.08720300e+0
2
    2.01400000e+03   0.00000000e+00]
 [  1.57000000e+02   1.95000000e+01   1.21526000e+01   1.08378700e+0
2
    2.01000000e+03   0.00000000e+00]
 [  1.57000000e+02   3.00000000e+01   1.58600000e+01   1.07653800e+0
2
    2.00900000e+03   0.00000000e+00]]
[[ 0.04692255]
 [ 0.02934475]
 [ 0.00674784]
 ...,
 [ 0.41992825]
 [ 0.46247512]
 [ 0.46689498]]
8569
8569
```

In [5]:

```
# calculate scaling values for input data

mean_vals = np.mean(X_data,axis=0)
range_vals = np.max(X_data,axis=0) - np.min(X_data,axis=0)
```

In [9]:

```
# set up neural network

INPUT_SHAPE = X_data[0].shape
print(u"Input shape is: {0}".format(INPUT_SHAPE))
DROPOUT_RATE = 0.15
DENSE_LAYER_SIZE = 128

def myNet():
    model = Sequential()
    model.add(Lambda(lambda x: x - mean_vals,input_shape = INPUT_SHAPE))   # pla
ceholder for normalization
    model.add(Dense(DENSE_LAYER_SIZE,activation='relu'))
    model.add(Dropout(DROPOUT_RATE))
    model.add(Dense(DENSE_LAYER_SIZE,activation='relu'))
    model.add(Dropout(DROPOUT_RATE))
    model.add(Dense(DENSE_LAYER_SIZE,activation='relu'))
    model.add(Dense(1))
    return model

model = myNet()
model.compile(loss='mean_squared_error',optimizer='adam',metrics=['mean_absolute
_error'])
print("Model contains {0} parameters.".format(model.count_params()))
print(model.summary())
```

```
Input shape is: (6,)
Model contains 34049 parameters.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda_3 (Lambda)            (None, 6)                 0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               896
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               16512
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 129
=================================================================
Total params: 34,049
Trainable params: 34,049
Non-trainable params: 0
_________________________________________________________________
None
```

In [10]:

```
# fit model

BATCH_SIZE = 64
NUM_EPOCHS = 128
early_stop = EarlyStopping(monitor='val_loss',min_delta=0.001,patience=32)
check_point = ModelCheckpoint(WEIGHTS_FILE,monitor='val_loss',save_best_only=Tru
e,mode='max')
history_object = model.fit(x=X_data, y=y_data,
                           batch_size = BATCH_SIZE,
                           epochs = NUM_EPOCHS,
                           verbose = 1,
                           callbacks = [early_stop,check_point],
                           validation_split = VALIDATION_FRACTION)

# reload model with best weights from training
model = myNet()
model.load_weights(WEIGHTS_FILE)
model.compile(loss='mean_squared_error',optimizer='adam',metrics=['mean_absolute
_error'])
print("Finished training; model reloaded with optimum weights.")
model.save(WEIGHTS_FILE)
```

```
Train on 6855 samples, validate on 1714 samples
Epoch 1/128
6855/6855 [==============================] - 1s 148us/step - loss: 1
14.4369 - mean_absolute_error: 3.7769 - val_loss: 3.2402 - val_mean_
absolute_error: 0.8218
Epoch 2/128
6855/6855 [==============================] - 1s 74us/step - loss: 20
.1228 - mean_absolute_error: 1.9398 - val_loss: 1.3747 - val_mean_ab
solute_error: 0.6320
Epoch 3/128
6855/6855 [==============================] - 1s 78us/step - loss: 7.
7407 - mean_absolute_error: 1.2016 - val_loss: 0.6293 - val_mean_abs
olute_error: 0.5408
Epoch 4/128
6855/6855 [==============================] - 1s 77us/step - loss: 7.
0240 - mean_absolute_error: 1.0554 - val_loss: 0.2429 - val_mean_abs
olute_error: 0.3327
Epoch 5/128
6855/6855 [==============================] - 1s 78us/step - loss: 3.
2501 - mean_absolute_error: 0.8389 - val_loss: 0.5816 - val_mean_abs
olute_error: 0.4220
Epoch 6/128
6855/6855 [==============================] - 1s 73us/step - loss: 2.
2883 - mean_absolute_error: 0.7260 - val_loss: 0.1865 - val_mean_abs
olute_error: 0.3000
Epoch 7/128
6855/6855 [==============================] - 1s 73us/step - loss: 2.
1698 - mean_absolute_error: 0.6740 - val_loss: 0.4389 - val_mean_abs
olute_error: 0.3618
```

```
Epoch 8/128
6855/6855 [==============================] - 1s 74us/step - loss: 1.
7815 - mean_absolute_error: 0.6200 - val_loss: 0.3584 - val_mean_abs
olute_error: 0.3555
Epoch 9/128
6855/6855 [==============================] - 0s 67us/step - loss: 1.
6054 - mean_absolute_error: 0.5657 - val_loss: 0.3719 - val_mean_abs
olute_error: 0.3657
Epoch 10/128
6855/6855 [==============================] - 1s 77us/step - loss: 1.
5256 - mean_absolute_error: 0.5457 - val_loss: 0.1796 - val_mean_abs
olute_error: 0.2860
Epoch 11/128
6855/6855 [==============================] - 1s 81us/step - loss: 0.
9885 - mean_absolute_error: 0.4782 - val_loss: 0.3013 - val_mean_abs
olute_error: 0.3249
Epoch 12/128
6855/6855 [==============================] - 0s 73us/step - loss: 0.
9233 - mean_absolute_error: 0.4612 - val_loss: 0.1798 - val_mean_abs
olute_error: 0.2878
Epoch 13/128
6855/6855 [==============================] - 0s 66us/step - loss: 0.
8753 - mean_absolute_error: 0.4530 - val_loss: 0.1124 - val_mean_abs
olute_error: 0.2656
Epoch 14/128
6855/6855 [==============================] - 1s 77us/step - loss: 0.
5310 - mean_absolute_error: 0.3890 - val_loss: 0.0920 - val_mean_abs
olute_error: 0.2378
Epoch 15/128
6855/6855 [==============================] - 1s 83us/step - loss: 0.
5966 - mean_absolute_error: 0.3825 - val_loss: 0.1926 - val_mean_abs
olute_error: 0.2805
Epoch 16/128
6855/6855 [==============================] - 1s 80us/step - loss: 0.
4950 - mean_absolute_error: 0.3657 - val_loss: 0.0735 - val_mean_abs
olute_error: 0.2255
Epoch 17/128
6855/6855 [==============================] - 1s 75us/step - loss: 0.
4385 - mean_absolute_error: 0.3537 - val_loss: 0.1095 - val_mean_abs
olute_error: 0.2615
Epoch 18/128
6855/6855 [==============================] - 1s 88us/step - loss: 0.
3948 - mean_absolute_error: 0.3332 - val_loss: 0.1518 - val_mean_abs
olute_error: 0.2626
Epoch 19/128
6855/6855 [==============================] - 1s 95us/step - loss: 0.
3311 - mean_absolute_error: 0.3197 - val_loss: 0.1052 - val_mean_abs
olute_error: 0.2358
Epoch 20/128
6855/6855 [==============================] - 1s 81us/step - loss: 0.
3156 - mean_absolute_error: 0.3168 - val_loss: 0.0680 - val_mean_abs
olute_error: 0.2082
Epoch 21/128
```

```
6855/6855 [==============================] - 1s 78us/step - loss: 0.
3840 - mean_absolute_error: 0.3080 - val_loss: 0.1658 - val_mean_abs
olute_error: 0.2511
Epoch 22/128
6855/6855 [==============================] - 1s 81us/step - loss: 0.
3782 - mean_absolute_error: 0.3240 - val_loss: 0.0949 - val_mean_abs
olute_error: 0.2318
Epoch 23/128
6855/6855 [==============================] - 1s 78us/step - loss: 0.
2716 - mean_absolute_error: 0.2948 - val_loss: 0.0829 - val_mean_abs
olute_error: 0.2132
Epoch 24/128
6855/6855 [==============================] - 1s 76us/step - loss: 0.
2109 - mean_absolute_error: 0.2753 - val_loss: 0.0642 - val_mean_abs
olute_error: 0.2014
Epoch 25/128
6855/6855 [==============================] - 0s 73us/step - loss: 0.
1809 - mean_absolute_error: 0.2659 - val_loss: 0.0638 - val_mean_abs
olute_error: 0.1976
Epoch 26/128
6855/6855 [==============================] - 1s 75us/step - loss: 0.
1806 - mean_absolute_error: 0.2641 - val_loss: 0.0623 - val_mean_abs
olute_error: 0.1903
Epoch 27/128
6855/6855 [==============================] - 1s 75us/step - loss: 0.
2145 - mean_absolute_error: 0.2650 - val_loss: 0.0641 - val_mean_abs
olute_error: 0.1999
Epoch 28/128
6855/6855 [==============================] - 1s 74us/step - loss: 0.
1701 - mean_absolute_error: 0.2625 - val_loss: 0.0633 - val_mean_abs
olute_error: 0.2014
Epoch 29/128
6855/6855 [==============================] - 1s 77us/step - loss: 0.
1641 - mean_absolute_error: 0.2528 - val_loss: 0.0656 - val_mean_abs
olute_error: 0.2034
Epoch 30/128
6855/6855 [==============================] - 1s 76us/step - loss: 0.
1418 - mean_absolute_error: 0.2491 - val_loss: 0.0625 - val_mean_abs
olute_error: 0.1972
Epoch 31/128
6855/6855 [==============================] - 1s 80us/step - loss: 0.
1218 - mean_absolute_error: 0.2374 - val_loss: 0.0626 - val_mean_abs
olute_error: 0.1930
Epoch 32/128
6855/6855 [==============================] - 1s 80us/step - loss: 0.
1265 - mean_absolute_error: 0.2388 - val_loss: 0.0662 - val_mean_abs
olute_error: 0.2055
Epoch 33/128
6855/6855 [==============================] - 1s 77us/step - loss: 0.
1345 - mean_absolute_error: 0.2413 - val_loss: 0.0789 - val_mean_abs
olute_error: 0.2135
Epoch 34/128
6855/6855 [==============================] - 1s 79us/step - loss: 0.
```

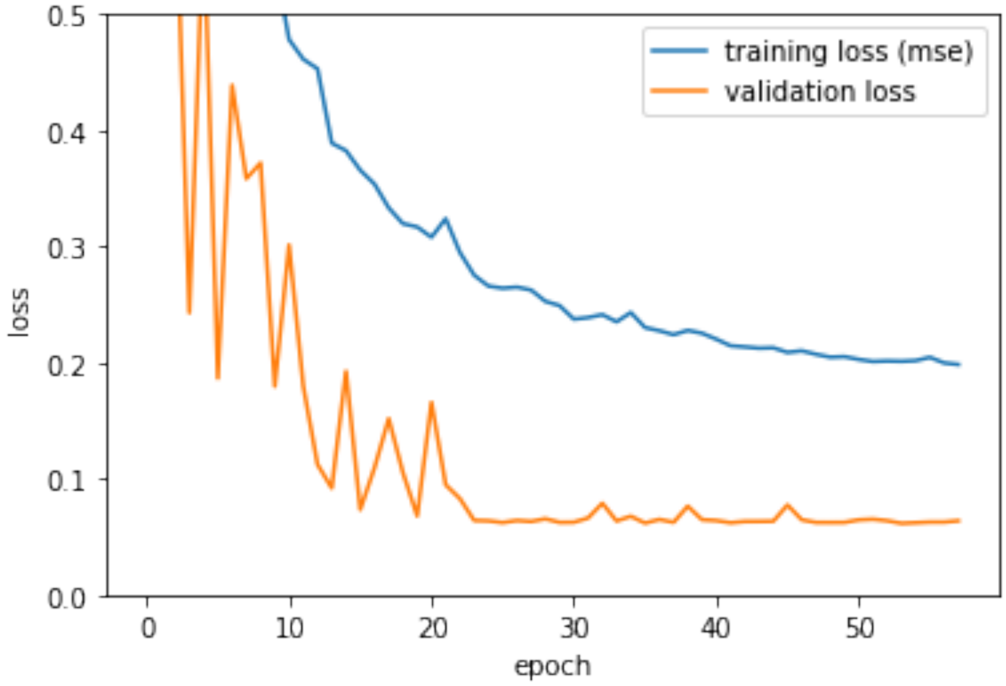
```
1251 - mean_absolute_error: 0.2352 - val_loss: 0.0637 - val_mean_abs
olute_error: 0.2006
Epoch 35/128
6855/6855 [==============================] - 1s 80us/step - loss: 0.
1502 - mean_absolute_error: 0.2431 - val_loss: 0.0678 - val_mean_abs
olute_error: 0.1999
Epoch 36/128
6855/6855 [==============================] - 1s 80us/step - loss: 0.
1117 - mean_absolute_error: 0.2302 - val_loss: 0.0619 - val_mean_abs
olute_error: 0.1942
Epoch 37/128
6855/6855 [==============================] - 1s 75us/step - loss: 0.
1240 - mean_absolute_error: 0.2275 - val_loss: 0.0650 - val_mean_abs
olute_error: 0.1971
Epoch 38/128
6855/6855 [==============================] - 1s 76us/step - loss: 0.
0970 - mean_absolute_error: 0.2241 - val_loss: 0.0625 - val_mean_abs
olute_error: 0.1968
Epoch 39/128
6855/6855 [==============================] - 1s 73us/step - loss: 0.
1294 - mean_absolute_error: 0.2278 - val_loss: 0.0764 - val_mean_abs
olute_error: 0.2117
Epoch 40/128
6855/6855 [==============================] - 1s 74us/step - loss: 0.
1140 - mean_absolute_error: 0.2253 - val_loss: 0.0648 - val_mean_abs
olute_error: 0.2023
Epoch 41/128
6855/6855 [==============================] - 0s 72us/step - loss: 0.
0967 - mean_absolute_error: 0.2202 - val_loss: 0.0641 - val_mean_abs
olute_error: 0.2019
Epoch 42/128
6855/6855 [==============================] - 1s 76us/step - loss: 0.
0839 - mean_absolute_error: 0.2145 - val_loss: 0.0622 - val_mean_abs
olute_error: 0.1926
Epoch 43/128
6855/6855 [==============================] - 1s 76us/step - loss: 0.
0851 - mean_absolute_error: 0.2135 - val_loss: 0.0633 - val_mean_abs
olute_error: 0.1953
Epoch 44/128
6855/6855 [==============================] - 1s 75us/step - loss: 0.
0853 - mean_absolute_error: 0.2124 - val_loss: 0.0633 - val_mean_abs
olute_error: 0.2004
Epoch 45/128
6855/6855 [==============================] - 0s 73us/step - loss: 0.
0842 - mean_absolute_error: 0.2128 - val_loss: 0.0635 - val_mean_abs
olute_error: 0.2003
Epoch 46/128
6855/6855 [==============================] - 0s 73us/step - loss: 0.
0766 - mean_absolute_error: 0.2088 - val_loss: 0.0775 - val_mean_abs
olute_error: 0.2097
Epoch 47/128
6855/6855 [==============================] - 1s 75us/step - loss: 0.
0779 - mean_absolute_error: 0.2102 - val_loss: 0.0647 - val_mean_abs
```

```
olute_error: 0.1997
Epoch 48/128
6855/6855 [==============================] - 1s 75us/step - loss: 0.
0730 - mean_absolute_error: 0.2072 - val_loss: 0.0624 - val_mean_abs
olute_error: 0.1963
Epoch 49/128
6855/6855 [==============================] - 1s 87us/step - loss: 0.
0713 - mean_absolute_error: 0.2047 - val_loss: 0.0624 - val_mean_abs
olute_error: 0.1969
Epoch 50/128
6855/6855 [==============================] - 0s 71us/step - loss: 0.
0742 - mean_absolute_error: 0.2051 - val_loss: 0.0625 - val_mean_abs
olute_error: 0.1971
Epoch 51/128
6855/6855 [==============================] - 0s 68us/step - loss: 0.
0696 - mean_absolute_error: 0.2028 - val_loss: 0.0647 - val_mean_abs
olute_error: 0.1991
Epoch 52/128
6855/6855 [==============================] - 1s 75us/step - loss: 0.
0667 - mean_absolute_error: 0.2010 - val_loss: 0.0652 - val_mean_abs
olute_error: 0.1995
Epoch 53/128
6855/6855 [==============================] - 1s 82us/step - loss: 0.
0688 - mean_absolute_error: 0.2014 - val_loss: 0.0639 - val_mean_abs
olute_error: 0.1989
Epoch 54/128
6855/6855 [==============================] - 1s 79us/step - loss: 0.
0660 - mean_absolute_error: 0.2012 - val_loss: 0.0616 - val_mean_abs
olute_error: 0.1963
Epoch 55/128
6855/6855 [==============================] - 1s 79us/step - loss: 0.
0699 - mean_absolute_error: 0.2017 - val_loss: 0.0622 - val_mean_abs
olute_error: 0.1960
Epoch 56/128
6855/6855 [==============================] - 1s 80us/step - loss: 0.
0769 - mean_absolute_error: 0.2046 - val_loss: 0.0628 - val_mean_abs
olute_error: 0.1966
Epoch 57/128
6855/6855 [==============================] - ETA: 0s - loss: 0.0666
- mean_absolute_error: 0.199 - 1s 85us/step - loss: 0.0665 - mean_ab
solute_error: 0.1997 - val_loss: 0.0628 - val_mean_absolute_error: 0
.1965
Epoch 58/128
6855/6855 [==============================] - 1s 81us/step - loss: 0.
0669 - mean_absolute_error: 0.1984 - val_loss: 0.0639 - val_mean_abs
olute_error: 0.1984
Finished training; model reloaded with optimum weights.
```

In [11]:

```
# plot training loss history

plt.plot(history_object.history['mean_absolute_error'])
plt.plot(history_object.history['val_loss'])
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['training loss (mse)','validation loss'],loc='upper right')
plt.ylim([0,0.5])
plt.show()
```

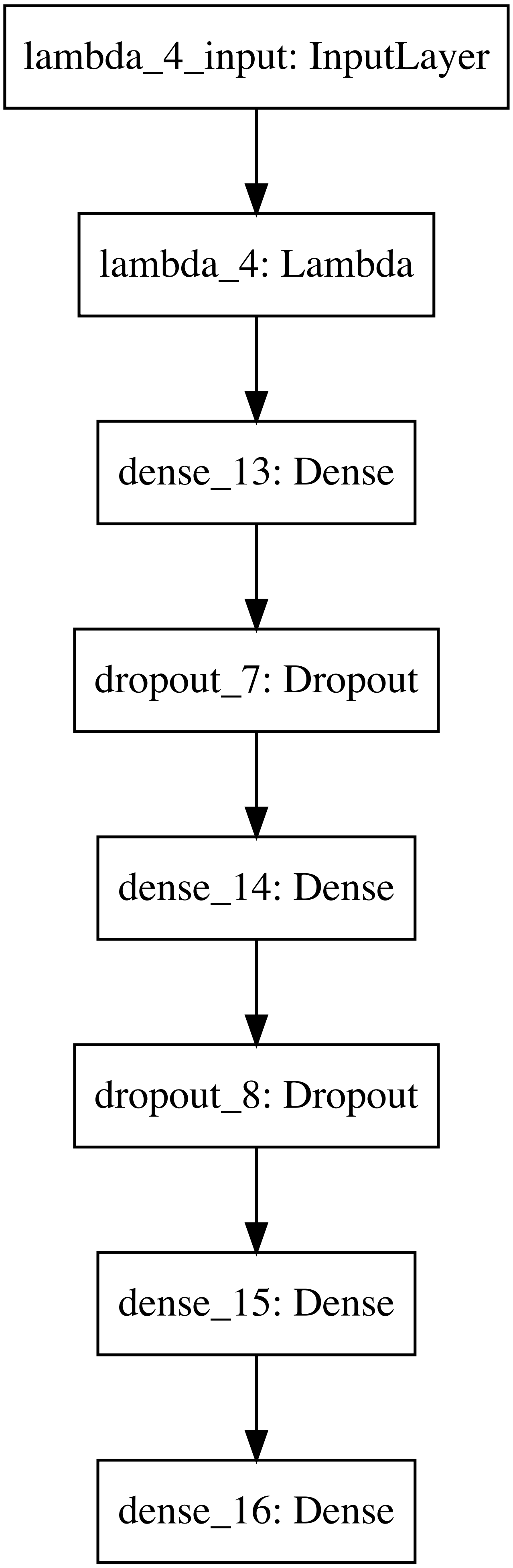


In [12]:

```
# visualize model

SVG(model_to_dot(model).create(prog='dot',format='svg'))
```

Out[12]:


lambda_4_input: InputLayer
lambda_4: Lambda
dense_13: Dense
dropout_7: Dropout
dense_14: Dense
dropout_8: Dropout
dense_15: Dense
dense_16: Dense