Project Notes

Samuel Navarro

June 22, 2019

Contents

1 Visualizing Loss

When calling `model.fit()` or `model.fit_generator()`, Keras outputs a history object that contains the training and validation loss for each epoch.

Code example in Listing 1:

2 Generators

Generators can be a great way to work with large amounts of data. Instead of storing the preprocessed data in memory all at once, using a generator you can pull pieces of the data and process them on the fly only when you need them, which is much more memory-efficient.

3 Useful Links

Behavioral Cloning with David Silver:
`https://www.youtube.com/watch?v=rpxZ87YFg0M&feature=youtube`

Drawbacks of Behavioral Cloning George:
`https://www.youtube.com/watch?v=Hxoke1lDJ9w`

function-in-keras:
`https://github.com/fchollet/keras/issues/3519`

```
from keras.models import Model
import matplotlib.pyplot as plt

history_object = model.fit_generator(train_generator, samples_per_epoch =
    len(train_samples), validation_data =
    validation_generator,
    nb_val_samples = len(validation_samples),
    nb_epoch=5, verbose=1)

### print the keys contained in the history object
print(history_object.history.keys())

### plot the training and validation loss for each epoch
plt.plot(history_object.history['loss'])
plt.plot(history_object.history['val_loss'])
plt.title('model mean squared error loss')
plt.ylabel('mean squared error loss')
plt.xlabel('epoch')
plt.legend(['training set', 'validation set'], loc='upper right')
plt.show()
```

Listing 1: Visualizing Loss

```
def fibonacci()
    numbers_list = []
    while 1:
        if(len(numbers_list) < 2):
            numbers_list.append(1)
        else:
            numbers_list.append(numbers_list[-1] + numbers_list[-2])
        yield numbers_list # change this line so it yields its list instead of 1

our_generator = fibonacci()
my_output = []

for i in range(10):
    my_output = (next(our_generator))

print(my_output)
```

Listing 2: Generators

Normalization:
https://keras.io/layers/normalization/
https://ustczen.gitbooks.io/keras/content/layers/normalization.html

BatchNormalization

Can check this activation methods links as well:

Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)
https://arxiv.org/pdf/1511.07289v1.pdf

Advanced Activations Layers
https://keras.io/layers/advanced_activations

Efficient BackProp
http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

Can check these links:
There are many methods to reduce overfitting i.e. - pooling layers, Batch Normalization layers, and L2 regularization etc..

https://pgaleone.eu/deep-learning/regularization/2017/01/10/anaysis-of-dropout/

Dropout

Dropout Analysis

You can watch Andrej Karpathy 's discussion on Deep Learning if needed:
https://www.youtube.com/watch?v=u6aEYuemt0M