CLML

Cosmology with Large scale structure using Machine Learning

2022.02.14

https://gitlab.sscc.uos.ac.kr/worldhsy/clml

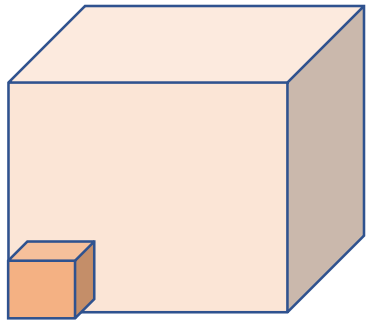
In the last meeting..

Brief review

- I had 6 data set

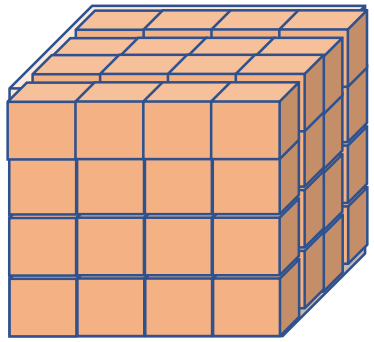
128_short	128_all
256_short	256_all
512_short	512_all

(short)



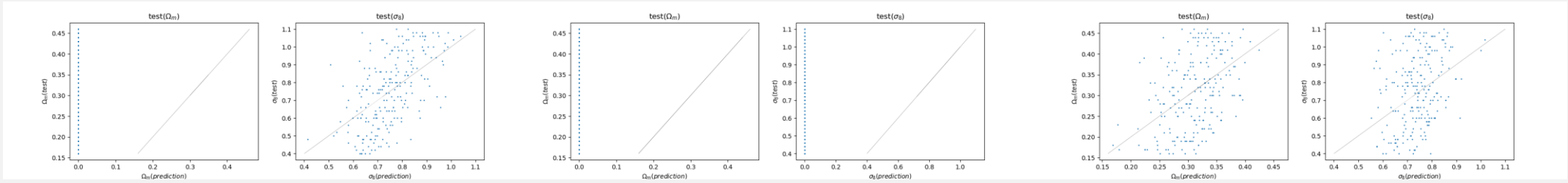
Only using [0,64](Mpc/h) scale

(all)



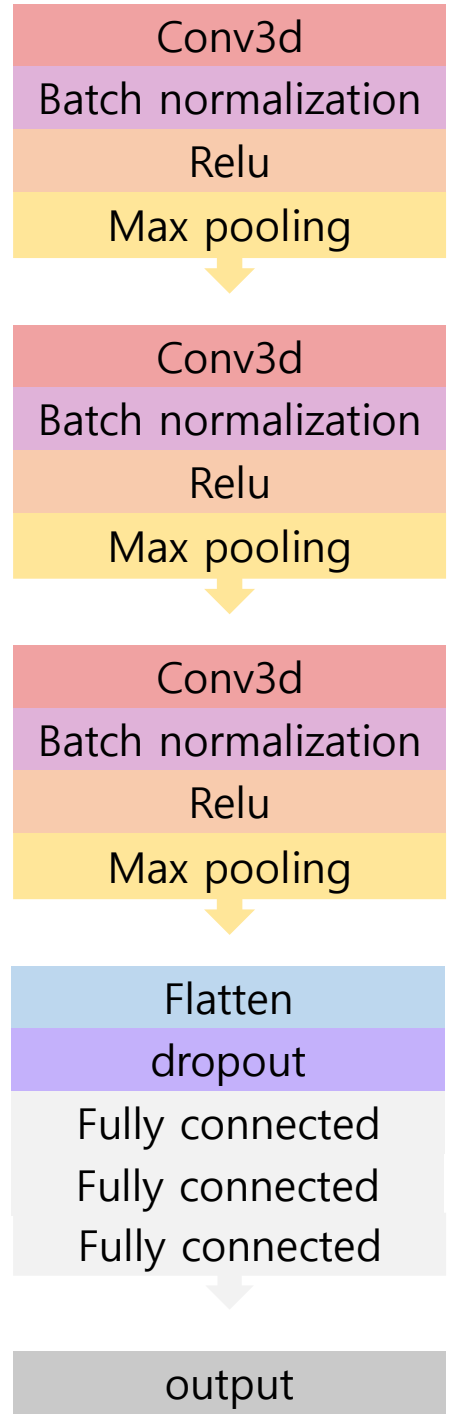
[0,64] [64,128] [128, 192] [192, 256]

- My model performance was not good



Basic model

```
def CNN():

    model = Sequential()

    model.add(Conv3D(32, kernel_size=(3, 3, 3), padding="valid", input_shape=(32,32,32,1)))
    model.add(BatchNormalization())
    model.add(ReLU())
    model.add(MaxPooling3D(pool_size=(2, 2, 2)))

    model.add(Conv3D(64, kernel_size=(3, 3, 3), padding="valid"))
    model.add(BatchNormalization())
    model.add(ReLU())
    model.add(MaxPooling3D(pool_size=(2, 2, 2)))

    model.add(Conv3D(128, kernel_size=(3, 3, 3), padding="valid"))
    model.add(BatchNormalization())
    model.add(ReLU())
    model.add(MaxPooling3D(pool_size=(2, 2, 2)))

    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Dense(1024, activation='relu'))
    model.add(Dense(256, activation='relu'))
    model.add(Dense(2, activation='linear'))

    return model
```

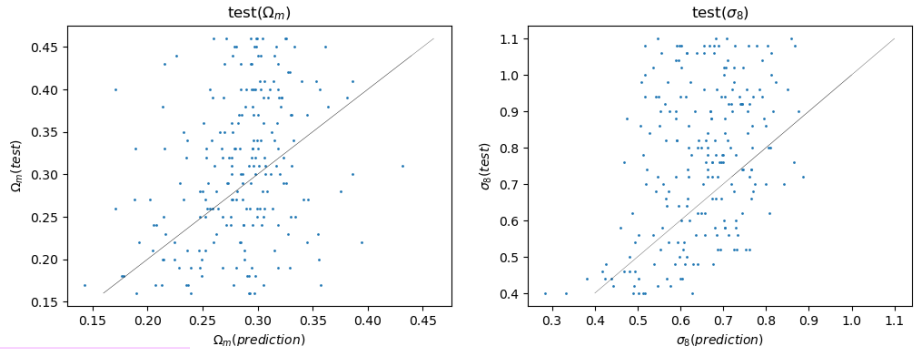


Callbacks.ModelCheckpoint

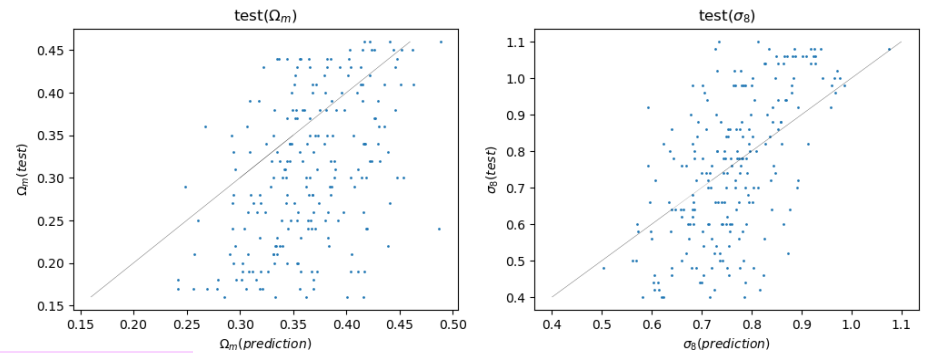
Different data

Epoch : 200
Learning rate : 0.0001

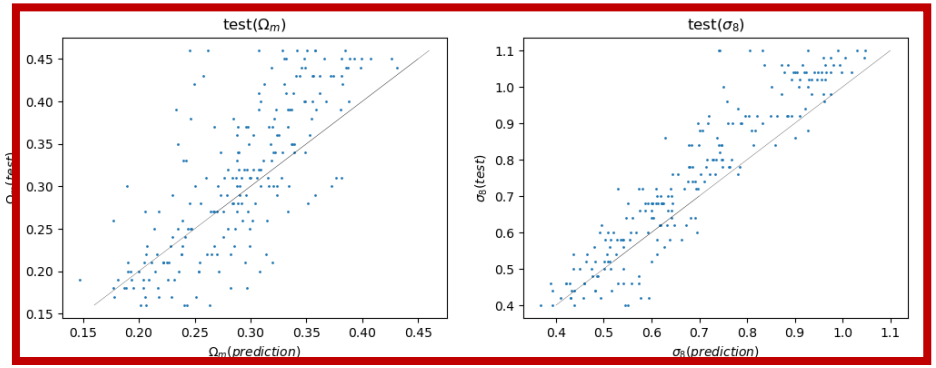
128_short



256_short



512_short



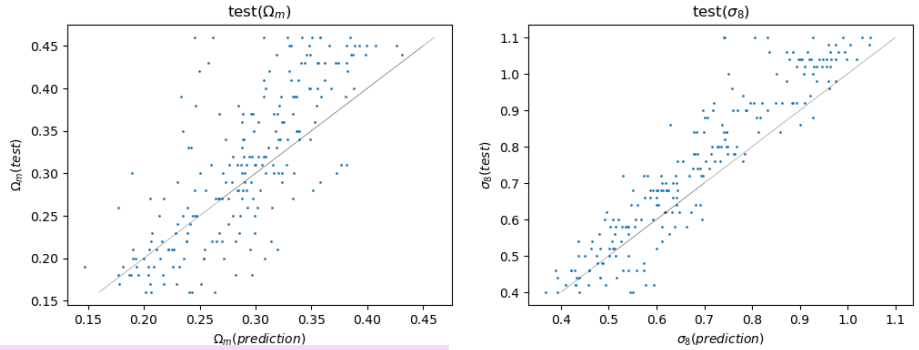
Different learning rate

Single cosmology

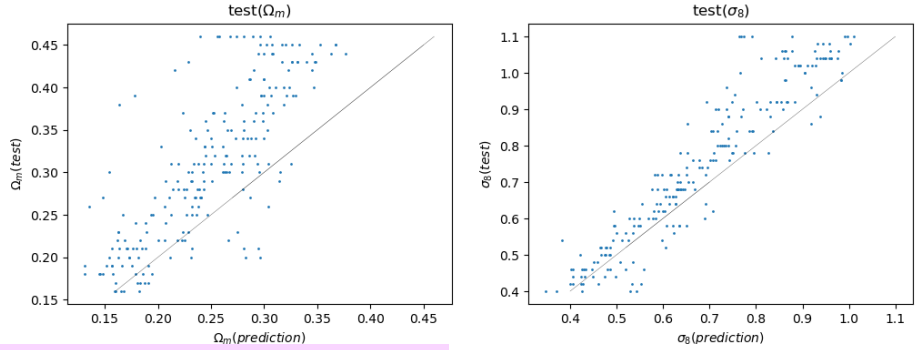
- Planck 2015 best fit cosmology

O_m = 0.3072
Sigma_8 = 0.8288
With different seed
Total number : 400

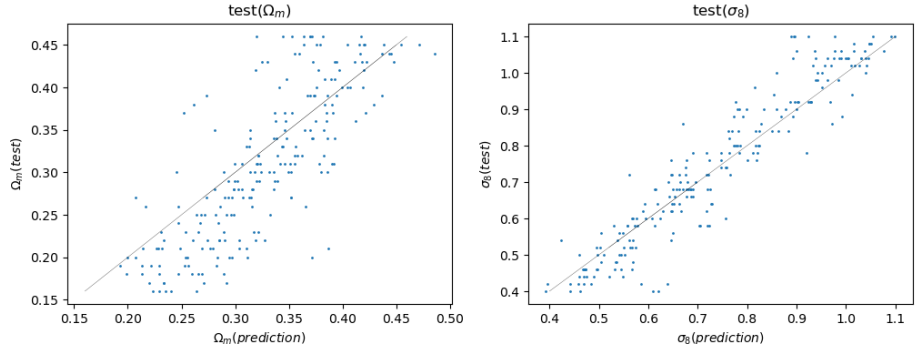
Learning rate : 0.0001



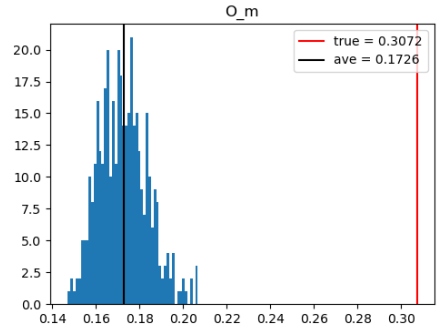
Learning rate : 0.0005

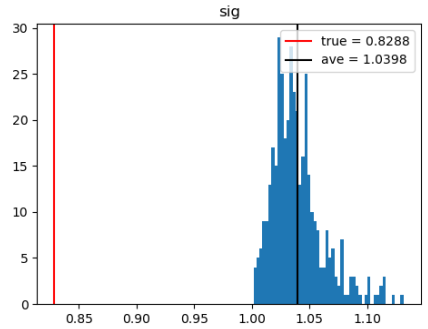


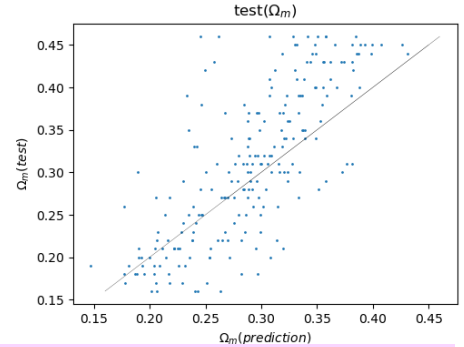
Learning rate : 0.001

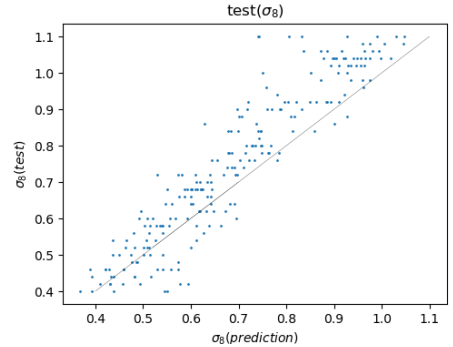


Learning rate : 0.0001

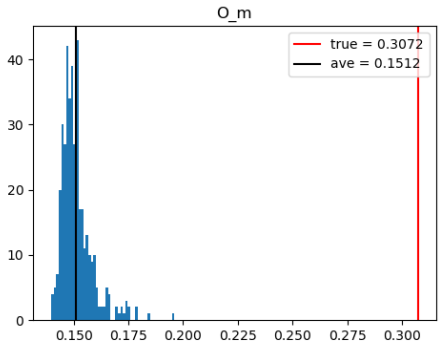


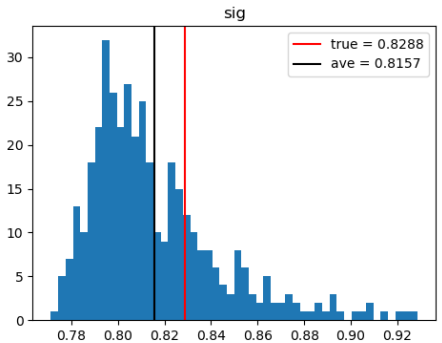


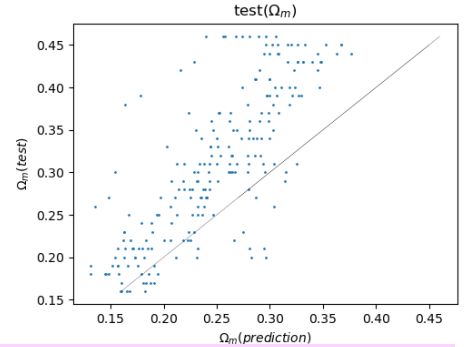


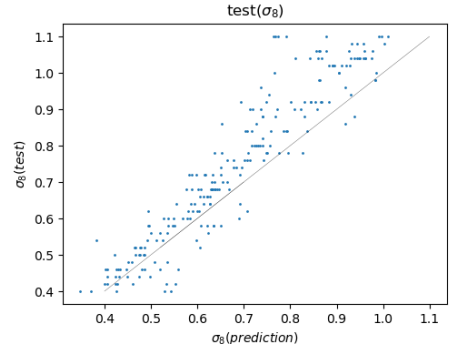


Learning rate : 0.0005

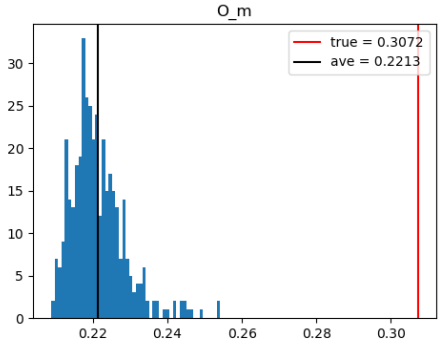


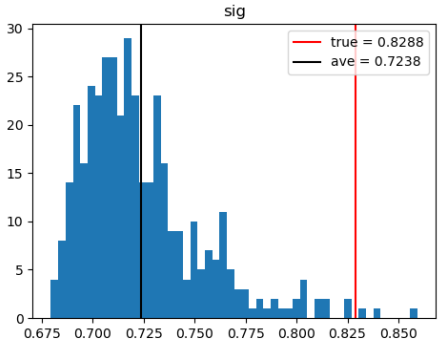


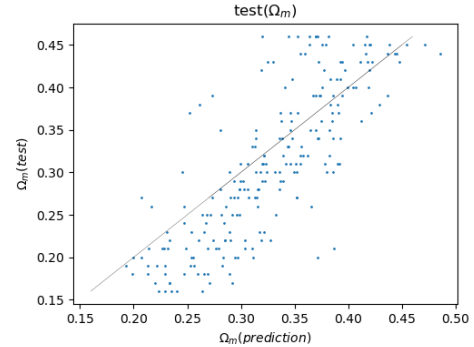


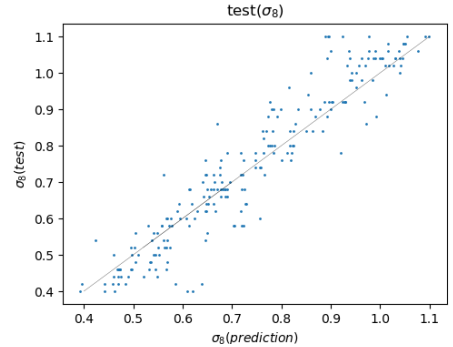


Learning rate : 0.001







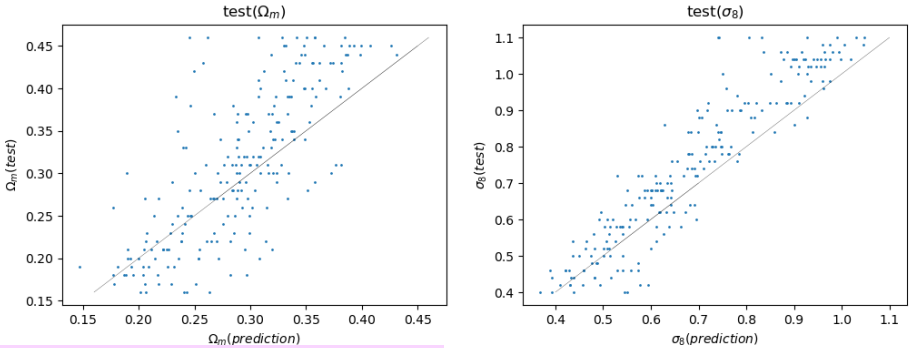


Put Averagepooling instead of Maxpooling

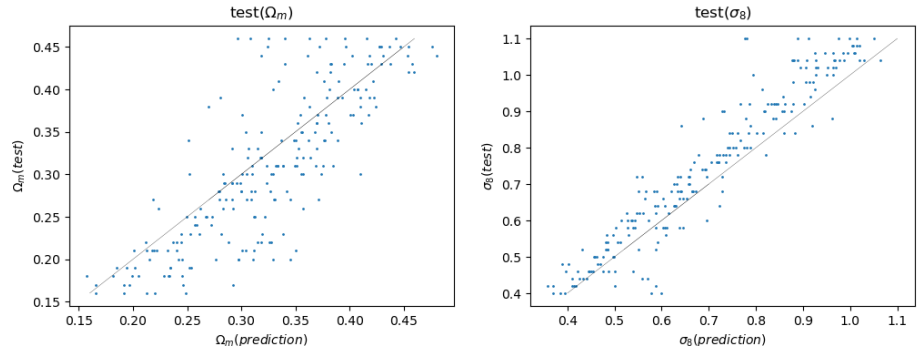
```
def CNN():

    model = Sequential()

    model.add(Conv3D(32, kernel_size=(3, 3, 3), padding="valid", input_shape=(32,32,32,1)))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling3D(pool_size=(2, 2, 2)))

    model.add(Conv3D(64, kernel_size=(3, 3, 3), padding="valid"))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling3D(pool_size=(2, 2, 2)))

    model.add(Conv3D(128, kernel_size=(3, 3, 3), padding="valid"))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling3D(pool_size=(2, 2, 2)))

    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Dense(1024))
    model.add(Activation('relu'))

    model.add(Dense(256))
    model.add(Activation('relu'))

    model.add(Dense(2))
    model.add(Activation('linear'))

    return model
```

Learning rate : 0.0001

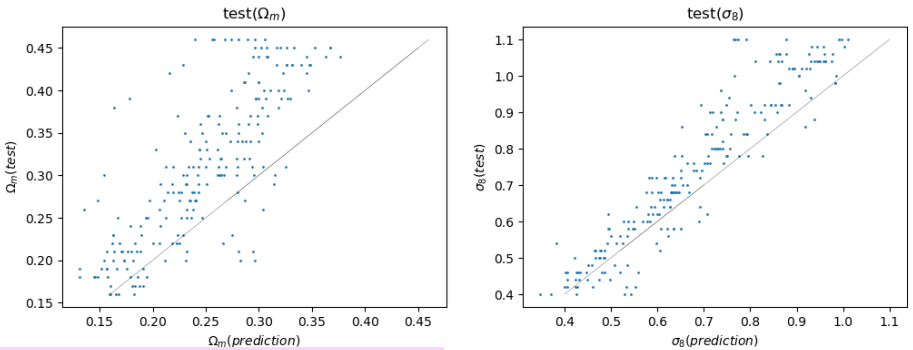
<basic>



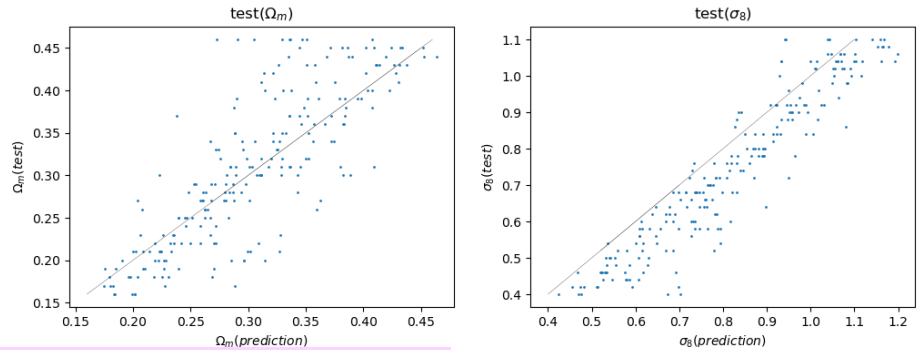
Learning rate : 0.0001



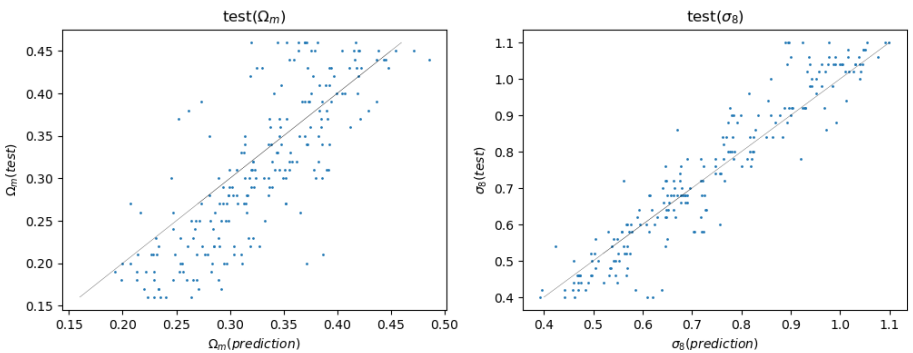
Learning rate : 0.0005



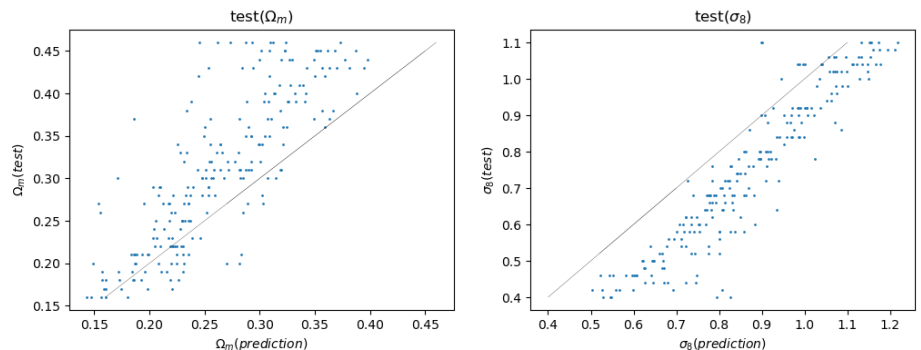
Learning rate : 0.0005



Learning rate : 0.001

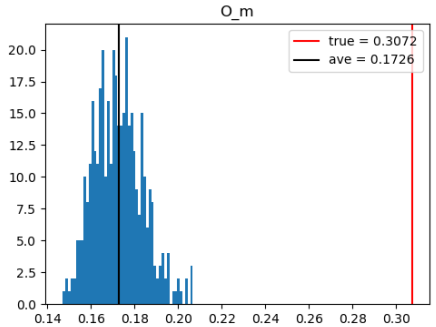


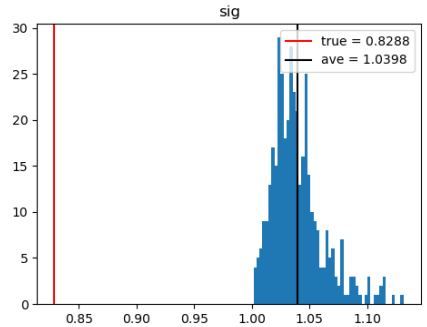
Learning rate : 0.001



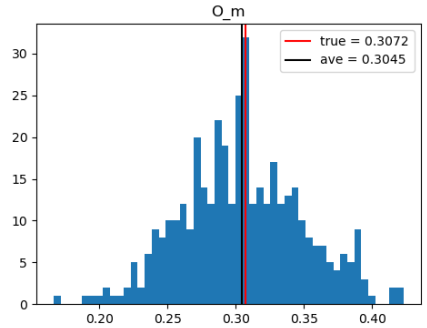
Learning rate : 0.0001

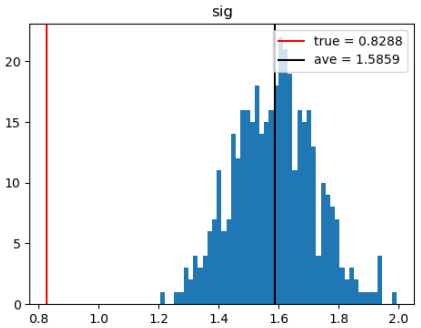
<basic>



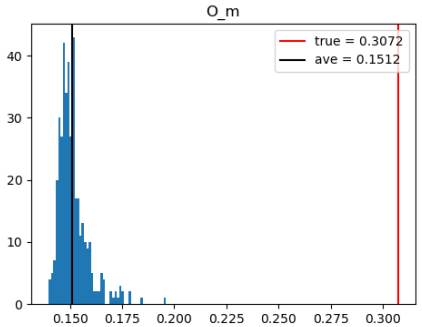


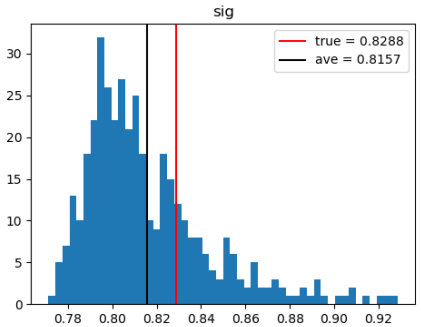
Learning rate : 0.0001



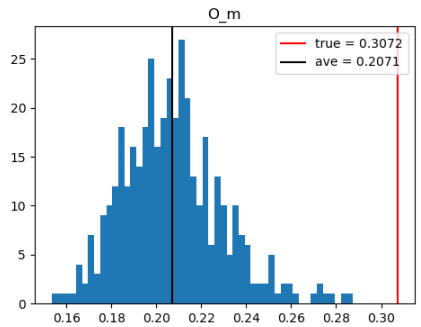


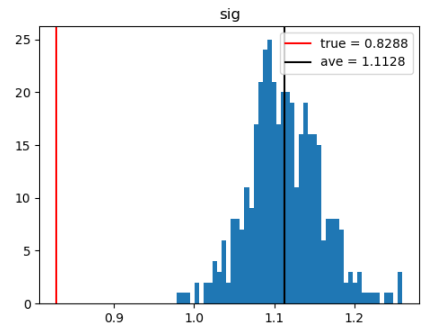
Learning rate : 0.0005



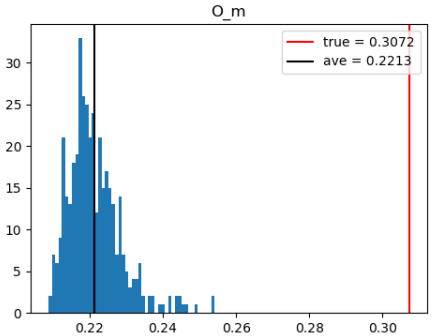


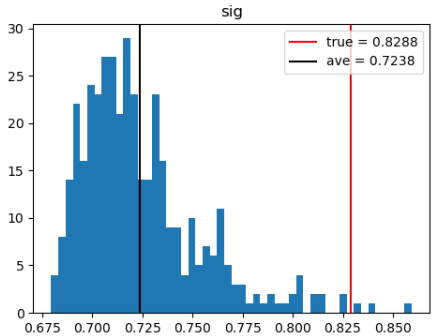
Learning rate : 0.0005



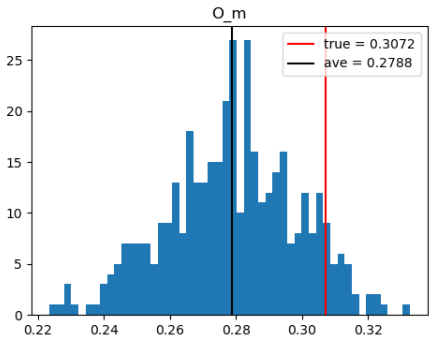


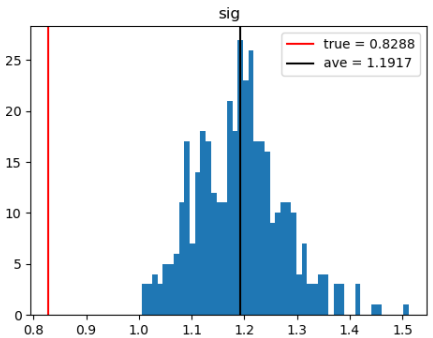
Learning rate : 0.001





Learning rate : 0.001



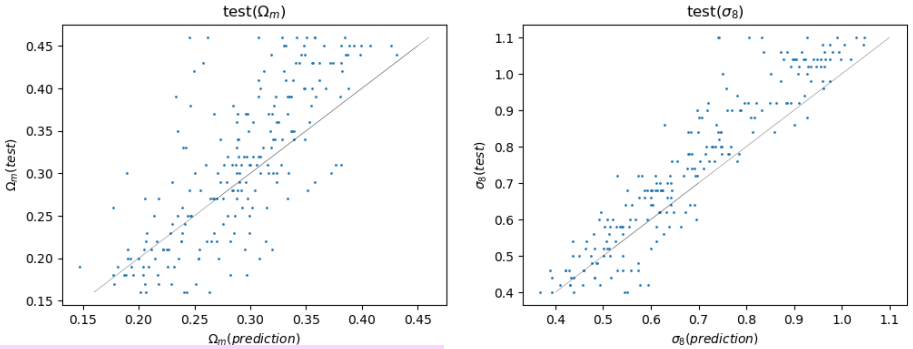


Subtract batchnormalization

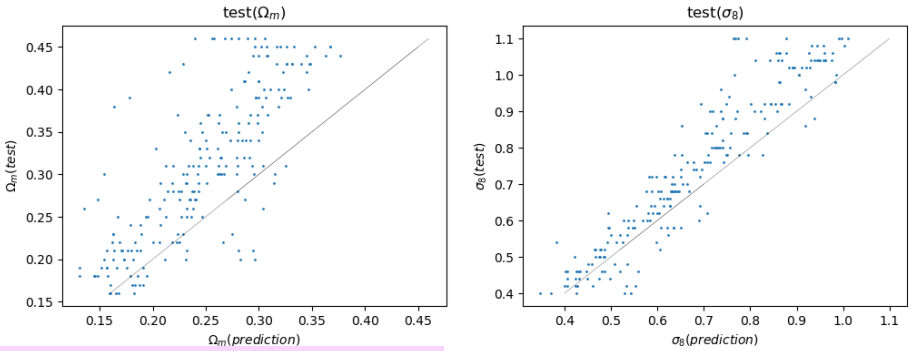
```
def CNN():

    model = Sequential()

    model.add(Conv3D(32, kernel_size=(3, 3, 3), padding="valid", input_shape=(32,32,32,1)))
#   model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling3D(pool_size=(2, 2, 2)))

    model.add(Conv3D(64, kernel_size=(3, 3, 3), padding="valid"))
#   model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling3D(pool_size=(2, 2, 2)))

    model.add(Conv3D(128, kernel_size=(3, 3, 3), padding="valid"))
#   model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling3D(pool_size=(2, 2, 2)))

    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Dense(1024))
    model.add(Activation('relu'))

    model.add(Dense(256))
    model.add(Activation('relu'))

    model.add(Dense(2))
    model.add(Activation('linear'))

    return model
```

Learning rate : 0.0001

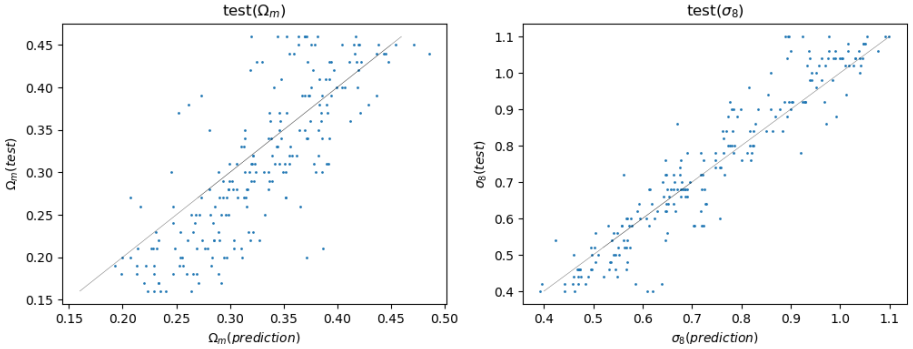
<basic>



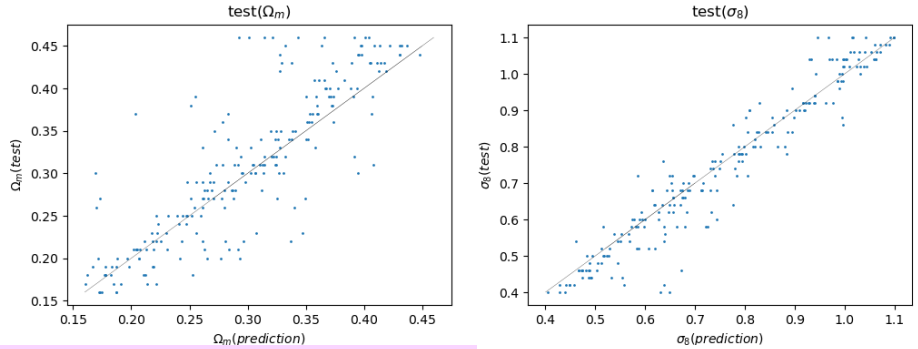
Learning rate : 0.0005



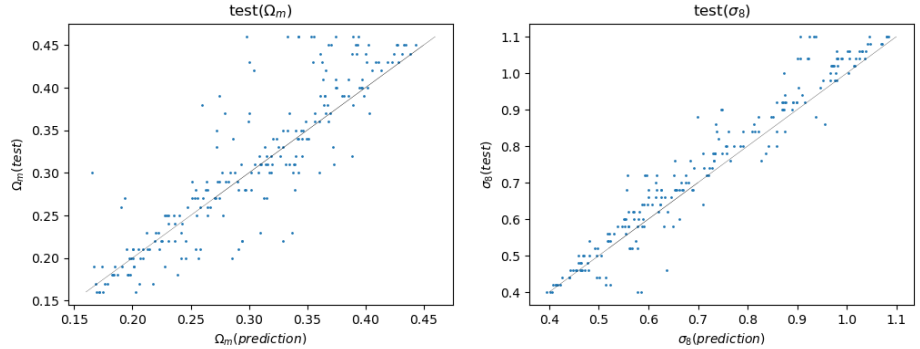
Learning rate : 0.001



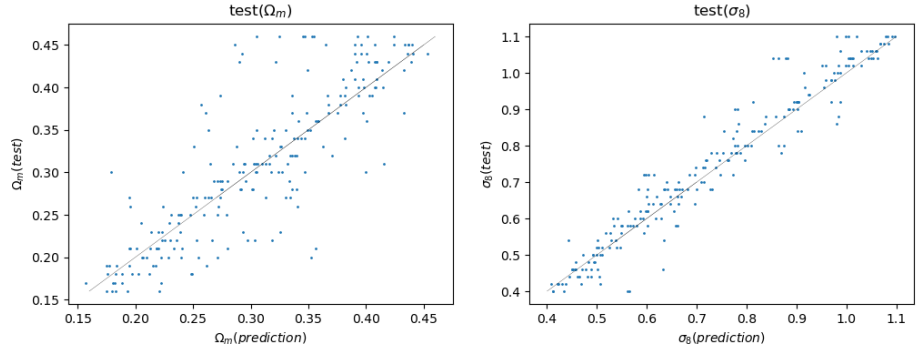
Learning rate : 0.0001



Learning rate : 0.0005

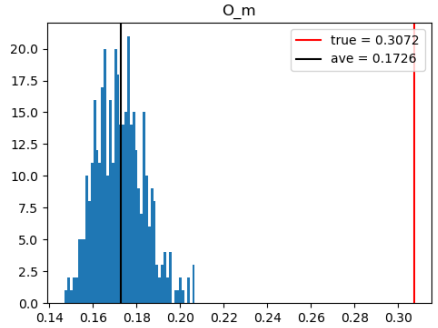


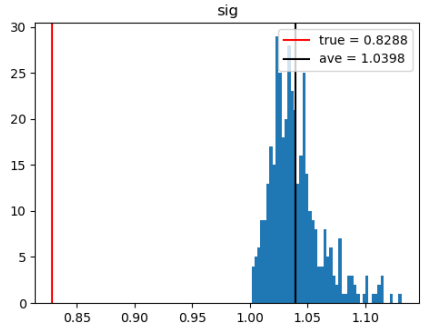
Learning rate : 0.001



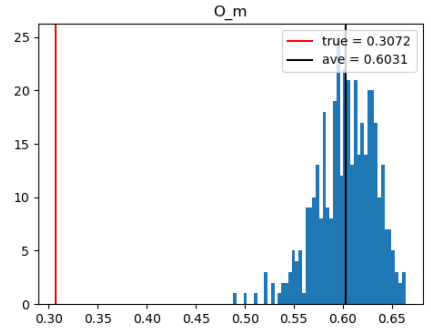
Learning rate : 0.0001

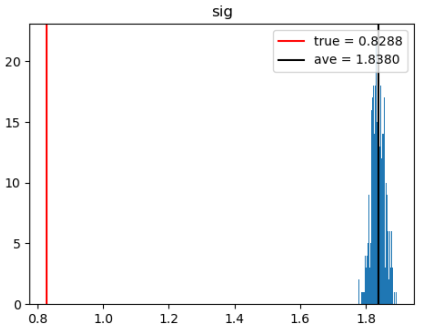
<basic>



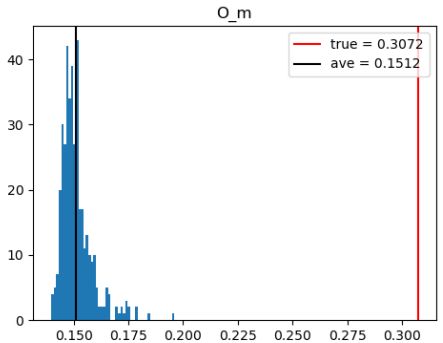


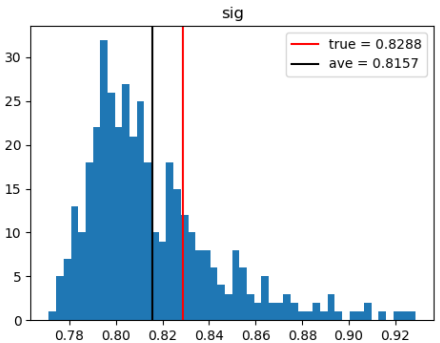
Learning rate : 0.0001



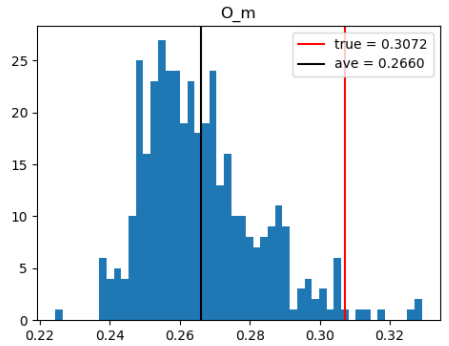


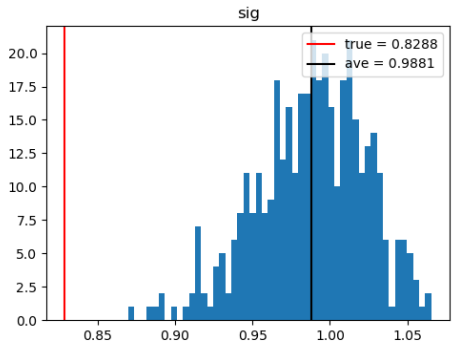
Learning rate : 0.0005



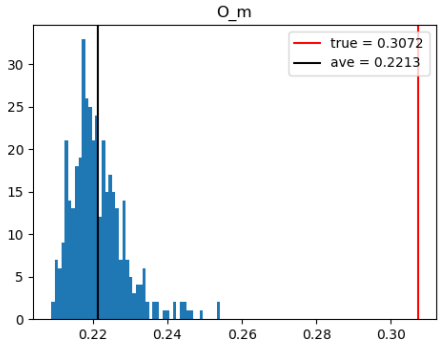


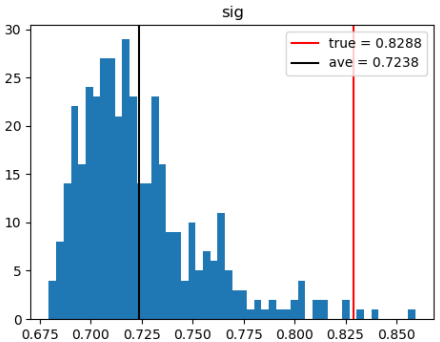
Learning rate : 0.0005



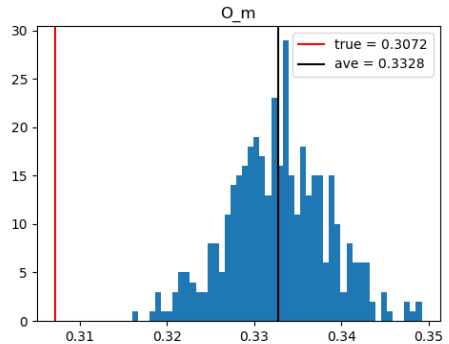


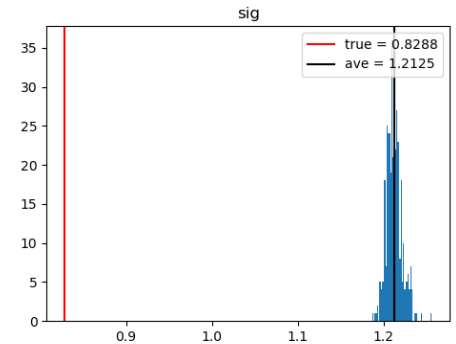
Learning rate : 0.001





Learning rate : 0.001



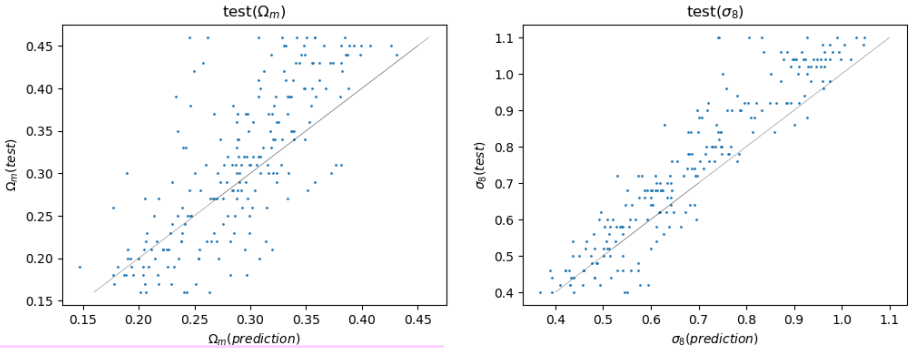


add batchnormalization in the full-connected layer

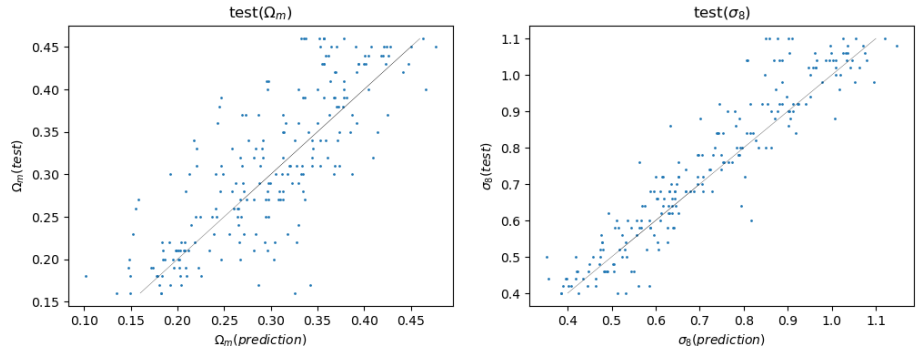
```
def CNN():

    model = Sequential()

    model.add(Conv3D(32, kernel_size=(3, 3, 3), padding="valid", input_shape=(32,32,32,1)))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling3D(pool_size=(2, 2, 2)))

    model.add(Conv3D(64, kernel_size=(3, 3, 3), padding="valid"))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling3D(pool_size=(2, 2, 2)))

    model.add(Conv3D(128, kernel_size=(3, 3, 3), padding="valid"))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling3D(pool_size=(2, 2, 2)))

    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Dense(1024))
    model.add(BatchNormalization())
    model.add(Activation('relu'))

    model.add(Dense(256))
    model.add(BatchNormalization())
    model.add(Activation('relu'))

    model.add(Dense(2))
    model.add(Activation('linear'))

    return model
```

Learning rate : 0.0001

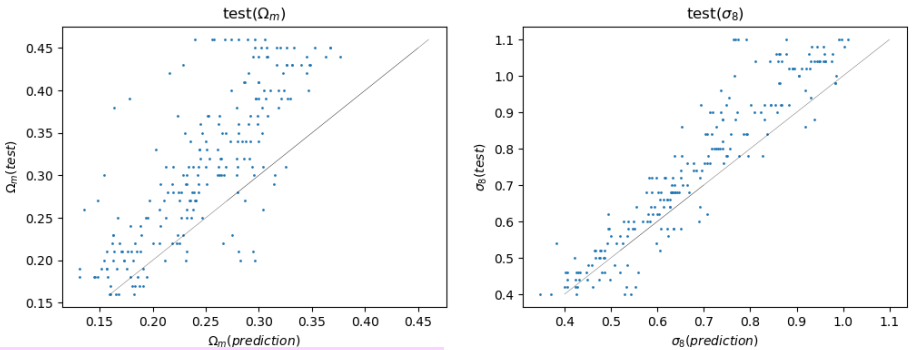
<basic>



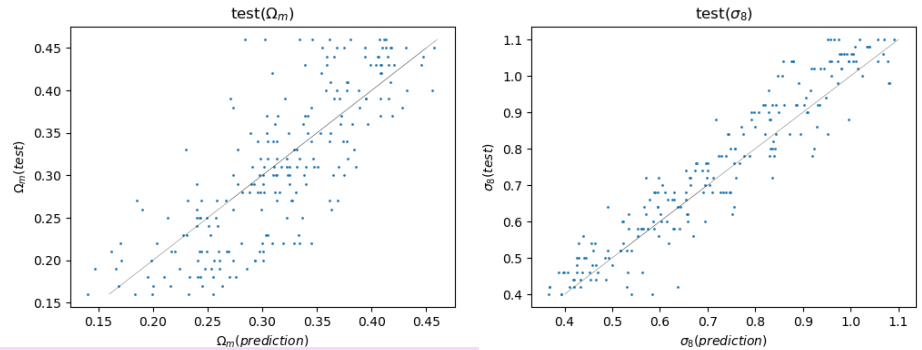
Learning rate : 0.0001



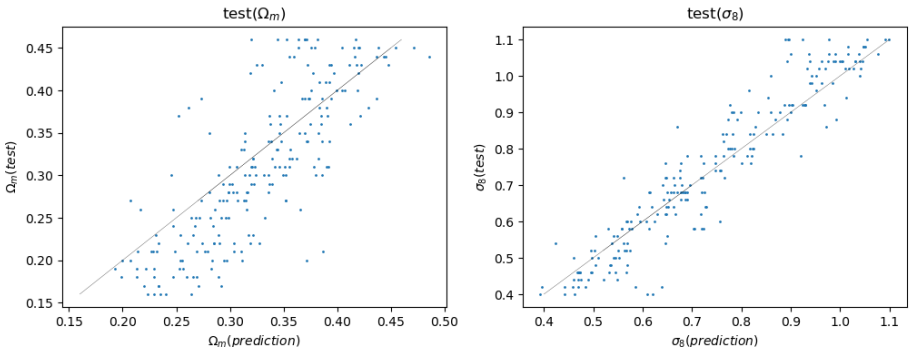
Learning rate : 0.0005



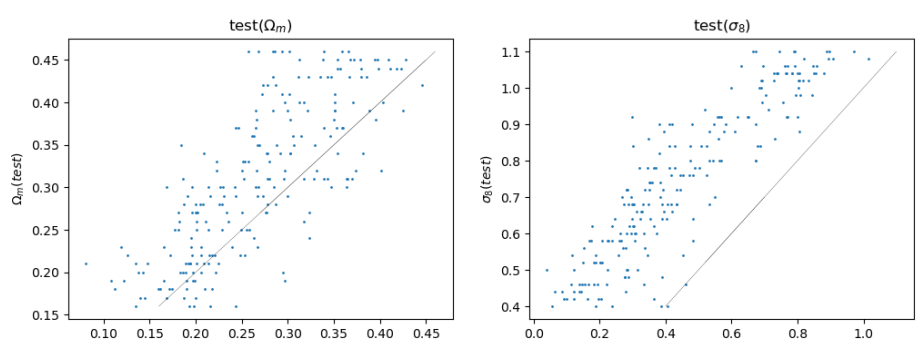
Learning rate : 0.0005



Learning rate : 0.001

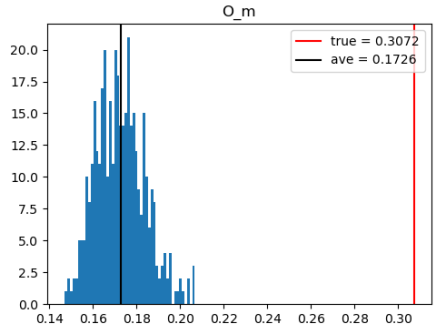


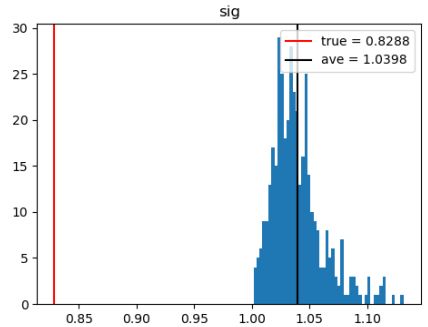
Learning rate : 0.001



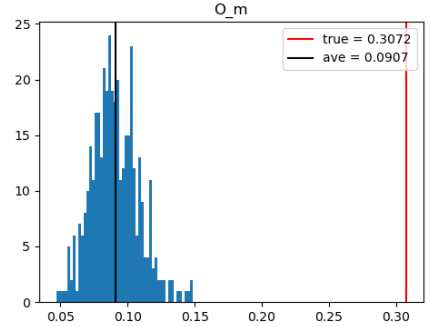
Learning rate : 0.0001

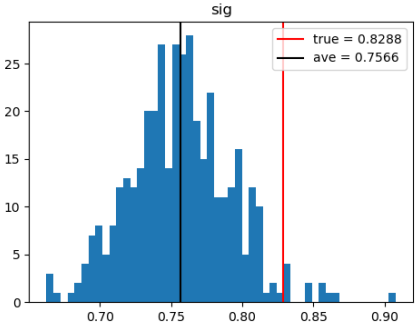
<basic>



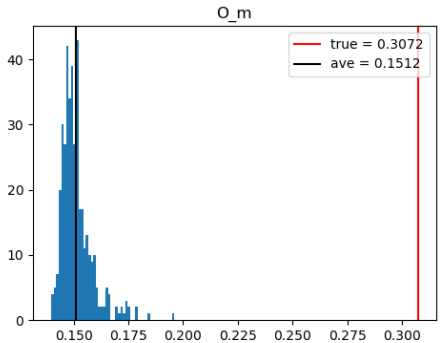


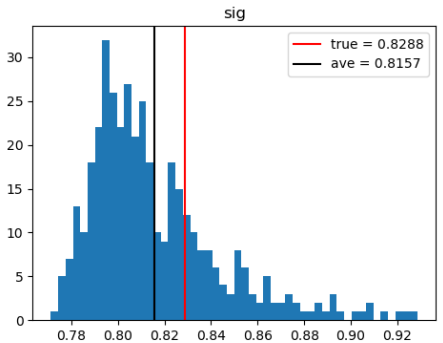
Learning rate : 0.0001



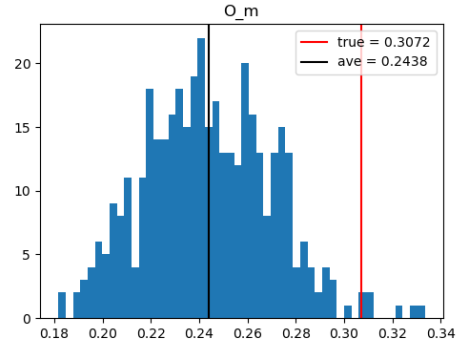


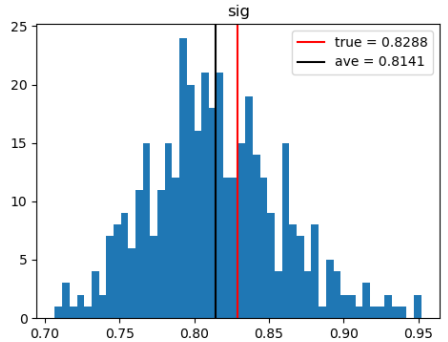
Learning rate : 0.0005



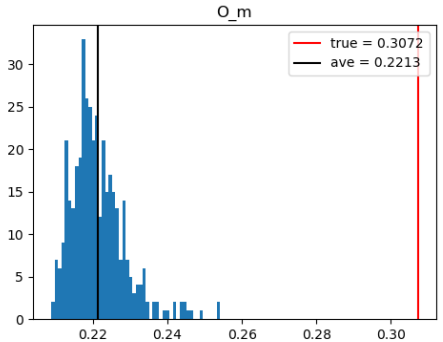


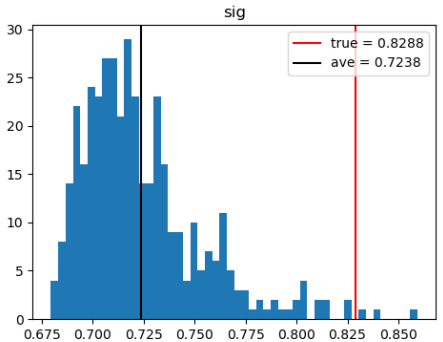
Learning rate : 0.0005



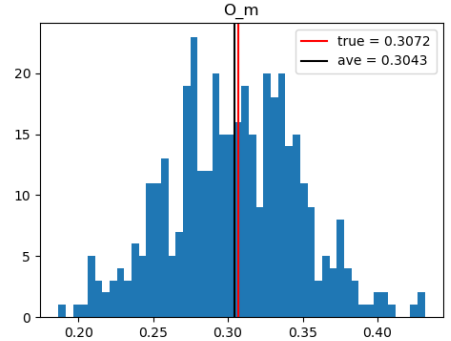


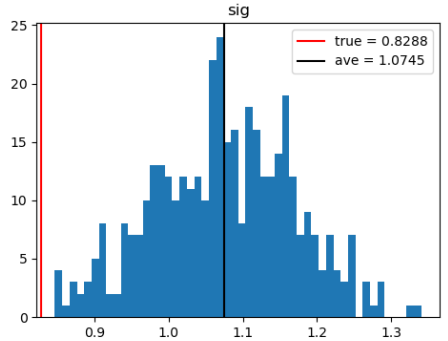
Learning rate : 0.001





Learning rate : 0.001

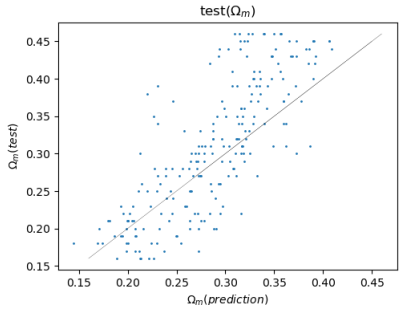


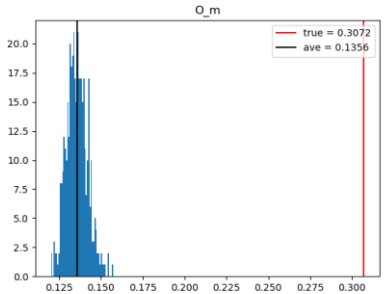


Let's predict single parameter!

Learning rate : 0.0001

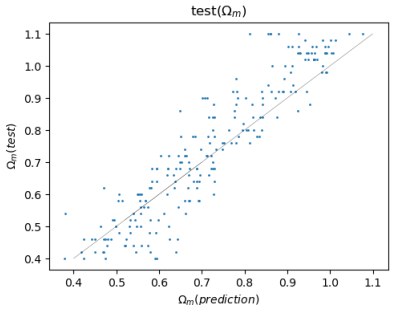
<Om>

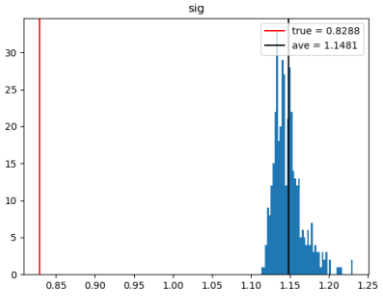




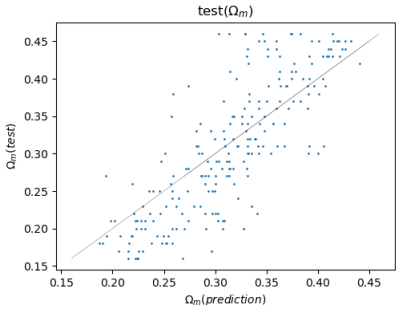
Learning rate : 0.0001

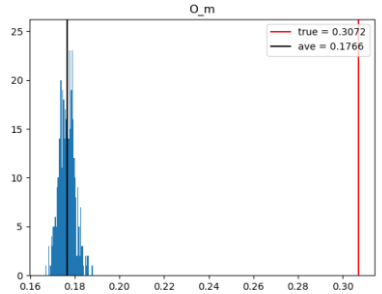
<sig>



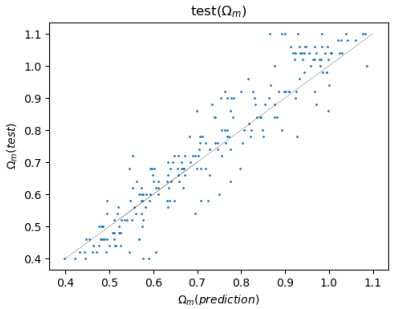


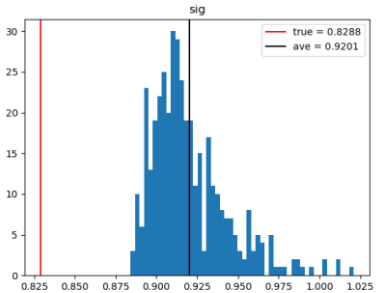
Learning rate : 0.0005



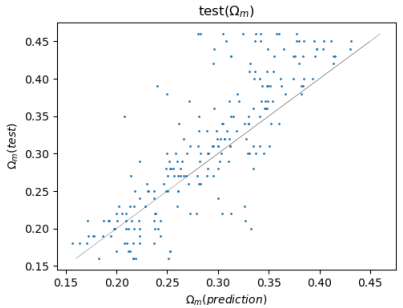


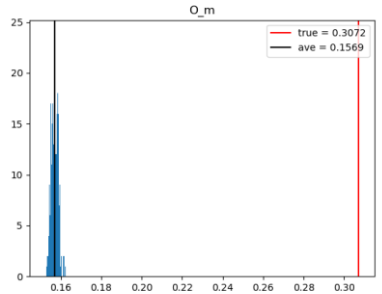
Learning rate : 0.0005



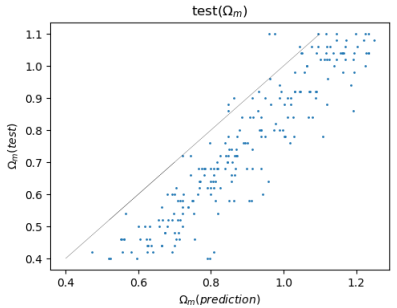


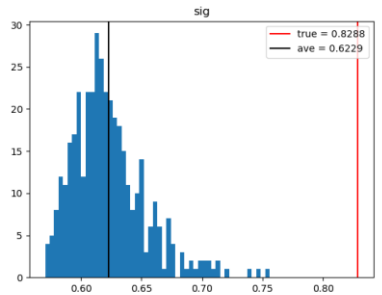
Learning rate : 0.001





Learning rate : 0.001





Next step...

- The difference between reference paper(1908.10590) and my work is:
  - They used "dark matter particles" and I used "dark matter halos"
  - It might be not fit for halo data
- I think I should more read about parameter estimation
- There are 2 useful paper(referenced by 1908.10590)

CosmoFlow: Using Deep Learning to Learn the Universe at Scale

Amrita Mathuriya*, Deborah Bard†, Peter Mendygral‡, Lawrence Meadows*, James Arnemann§,
Lei Shao¶, Siyu He**†‖, Tuomas Kärnä*, Diana Moise‡, Simon J. Pennycook¶,
Kristyn Maschhoff‡, Jason Sewall¶, Nalini Kumar¶, Shirley Ho**†‖, Michael F. Ringenburg‡, Prabhat† and Victor Lee¶
*Intel Corporation, 2111 NE 25th Ave, JF5, Hillsboro, OR 97124, USA. Email: amrita.mathuriya@intel.com
†Lawrence Berkeley National Laboratory, 1 Cyclotron Road, M/S 59R4010A, Berkeley, CA 94720, USA
‡Cray Inc., 901 Fifth Avenue, Suite 1000, Seattle, WA 98164, USA
§ U.C. Berkeley, Berkeley, CCA 94720, USA
¶Intel Corporation, 2200 Mission College Blvd., Santa Clara, CA 95054, USA
‖McWilliams Center for Cosmology, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA
**Center for Computational Astrophysics, Flatiron Institute, 162 5th Ave, New York, NY 10010, USA

***Abstract*—Deep learning is a promising tool to determine the physical model that describes our universe. To handle the considerable computational cost of this problem, we present CosmoFlow: a highly scalable deep learning application built on top of the TensorFlow framework. CosmoFlow uses efficient implementations of 3D convolution and pooling primitives, together with improvements in threading for many element-wise operations, to improve training performance on Intel® Xeon**

B. Deep Learning for Cosmology

The nature of dark energy is one of the most exciting and fundamental questions facing scientists today. Dark energy is the unknown force that is driving the accelerated expansion of the universe, and is the subject of several current and future experiments that will survey the sky in multiple wavelengths

Estimating Cosmological Parameters from the Dark Matter Distribution

Siamak Ravanbakhsh* MRAVANBA@CS.CMU.EDU
Junier Oliva* JOLIVA@CS.CMU.EDU
Sebastien Fromenteau† SFROMENT@ANDREW.CMU.EDU
Layne C. Price† LAYNEP@ANDREW.CMU.EDU
Shirley Ho† SHIRLEYH@ANDREW.CMU.EDU
Jeff Schneider* JEFF.SCHNEIDER@CS.CMU.EDU
Barnabás Póczos* BAPOCZOS@CS.CMU.EDU
⋆ School of Computer Science, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, USA
† McWilliams Center for Cosmology, Department of Physics, Carnegie Mellon University, Carnegie 5000 Forbes Ave., Pittsburgh, PA 15213, USA

Abstract

A grand challenge of the 21*st* century cosmology is to accurately estimate the cosmological parameters of our Universe. A major approach in estimating the cosmological parameters is to use the large scale matter distribution of the Universe. Galaxy surveys provide the means to map out cosmic large-scale structure in three dimensions. Information about galaxy locations is typically summarized in a "single" function of scale, such as the galaxy correlation function or power-spectrum. We show that it is possible to estimate these cosmological parameters directly from the distribution of matter. This paper presents the application of deep 3D convolutional networks to volumetric representation of dark-matter simulations as well as the results obtained using a recently proposed distribution regression frame-

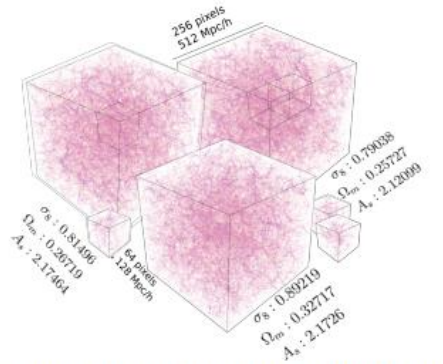


Figure 1. Dark matter distribution in three cubes produced using different sets of parameters. Each cube is divided into small sub-