Practice 04

MLP and Regularization

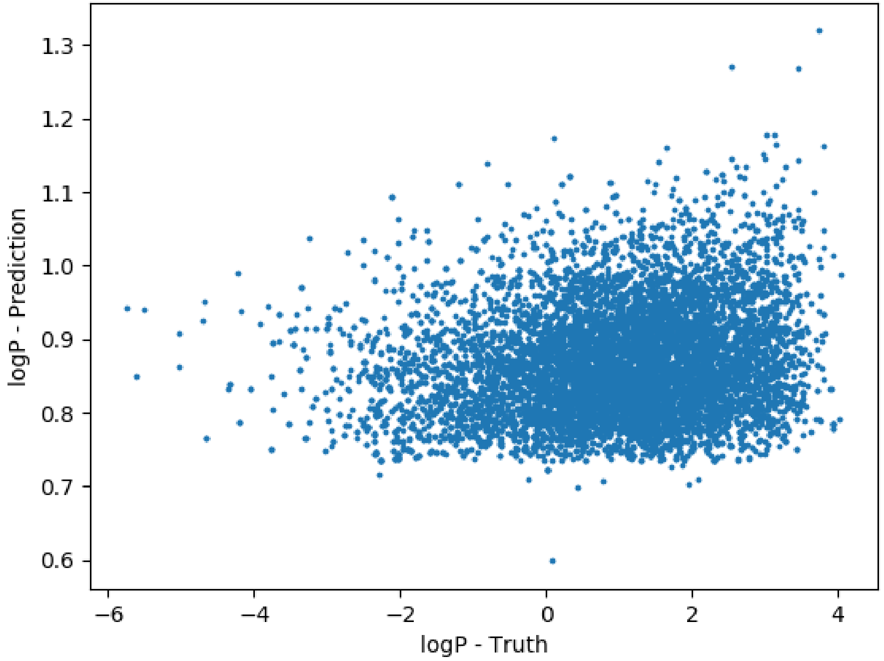
Seongok Ryu
Department of Chemistry, KAIST

Contents

- Results : logP prediction using MLP
- How can we improve the models?
- Regularization
- Assignment #3

Multi-layer perceptron

Test result

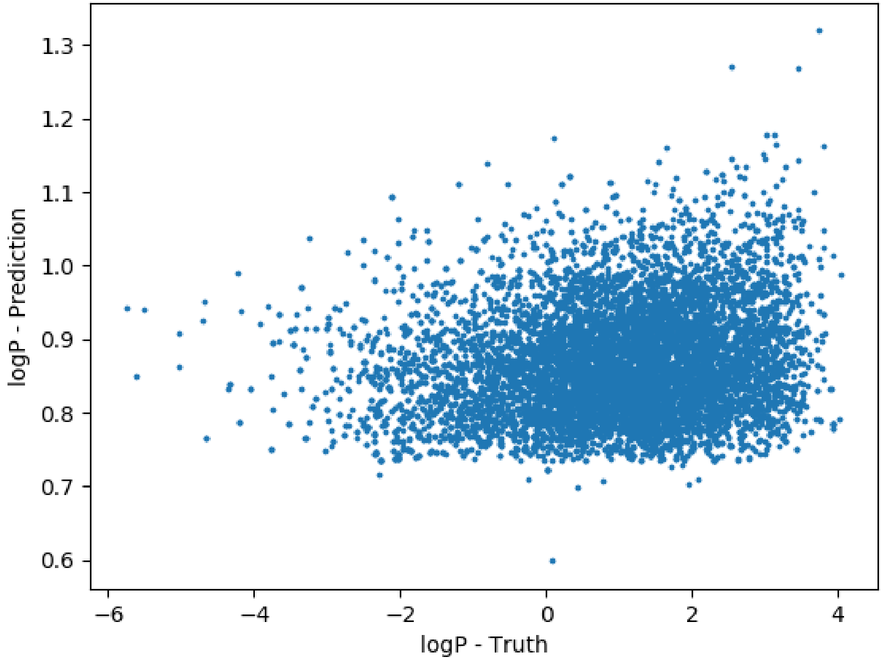


- Batch size = 100
- Epoch size = 100
- Learning rate = 0.001
- Decay rate = 0.95
- # Train = 40,000 / # Validation = 10,000 / # Test = 10,000

Totally wrong result! WHY?

Multi-layer perceptron

Test result



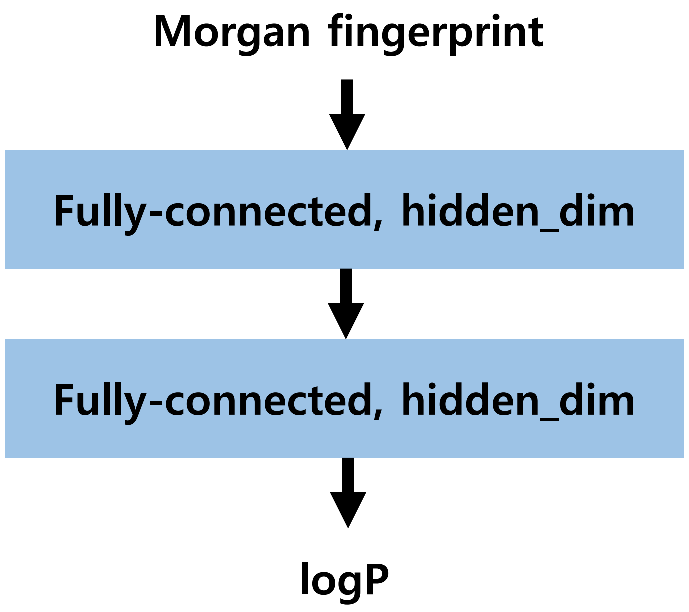
- Batch size = 100
- Epoch size = 100
- Learning rate = 0.001
- Decay rate = 0.95
- # Train = 40,000 / # Validation = 10,000 / # Test = 10,000

Totally wrong result! WHY?

It was due to my mistake T_T…

Results : logP prediction using MLP

I did a grid search of [num_layers, hidden_dim, learning_rate] to obtain the best MLP model.

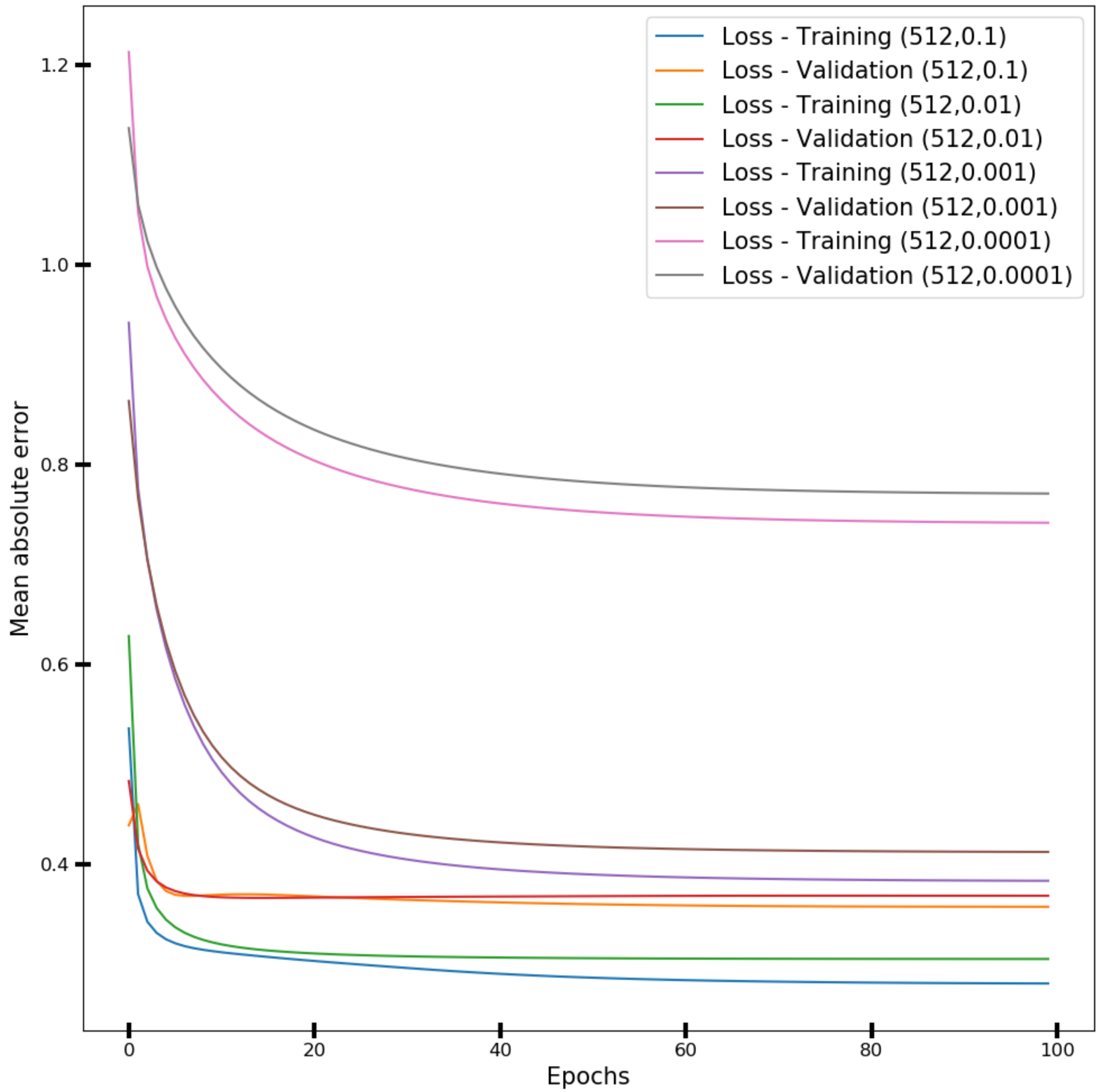


- num_layers : number of hidden layers
  → [1,2,3]
- hidden_dim : dimension of each hidden layer
  → [512, 1024, 2048]
- learning_rate : initial learning rate for training
  → [0.1, 0.01, 0.001, 0.0001]
- Total number of models : 3 x 3 x 4 = 36

Results : logP prediction using MLP

num_layer = 1, hidden_dim = 512

Learning rate	Test Accuracy
0.1	0.3549
0.01	0.3616
0.001	0.4159
0.0001	0.7601

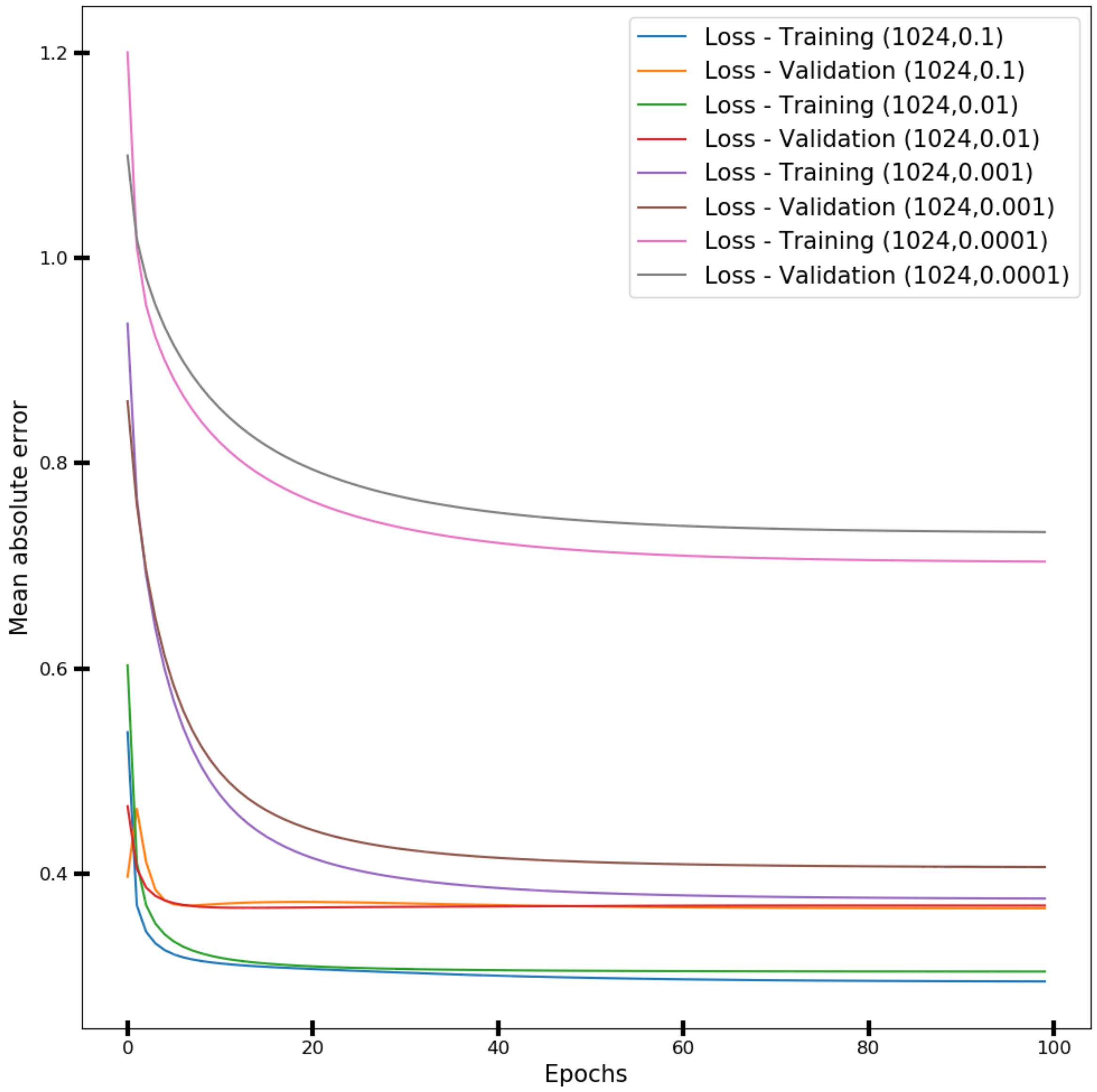


KAIST

Results : logP prediction using MLP

num_layer = 1, hidden_dim = 1024

Learning rate	Test Accuracy
0.1	0.3607
0.01	0.3610
0.001	0.4033
0.0001	0.7243

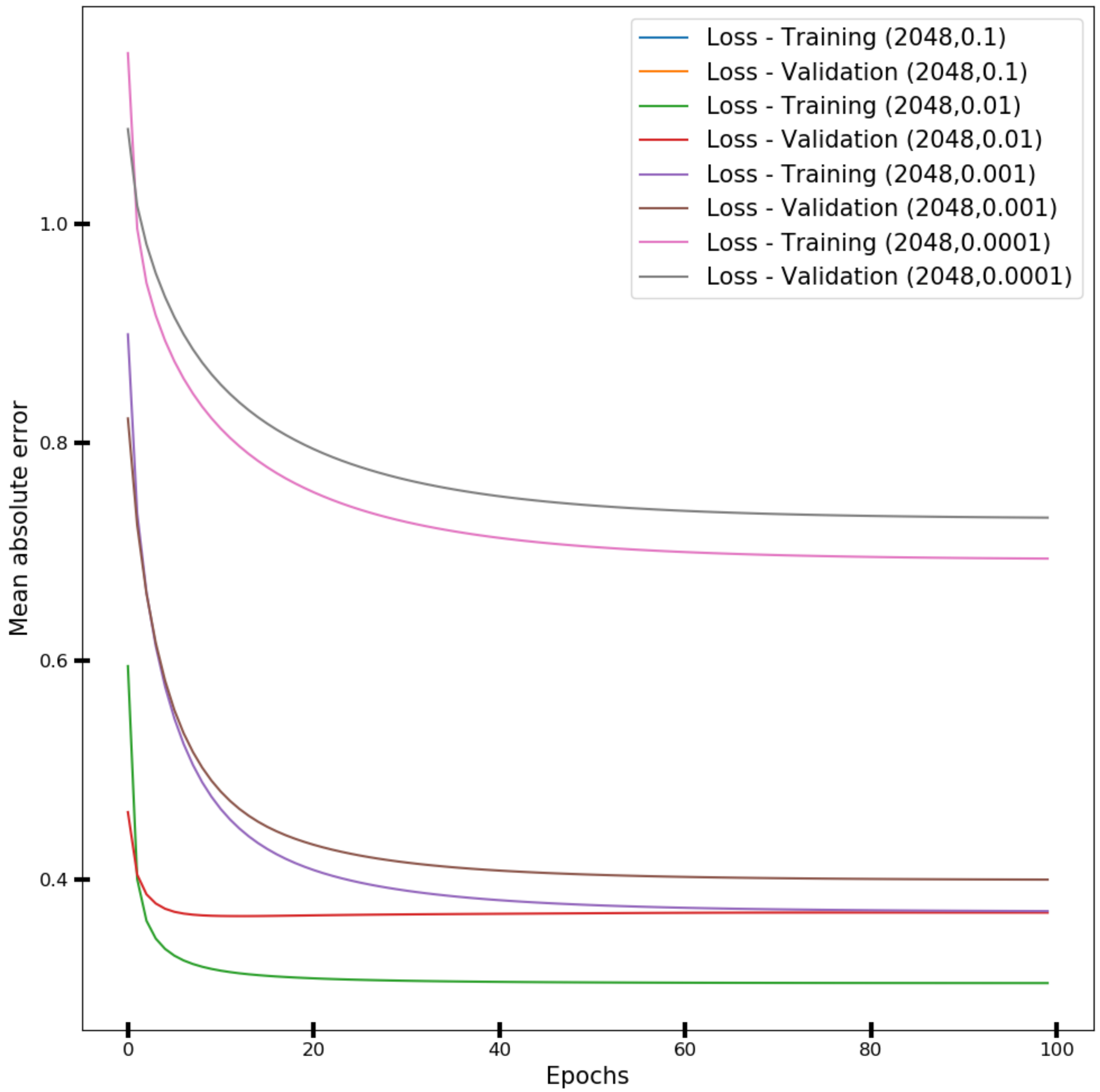


KAIST

Results : logP prediction using MLP

num_layer = 1, hidden_dim = 2048

Learning rate	Test Accuracy
0.1	Fail
0.01	**0.3621**
0.001	0.4002
0.0001	**0.7141**

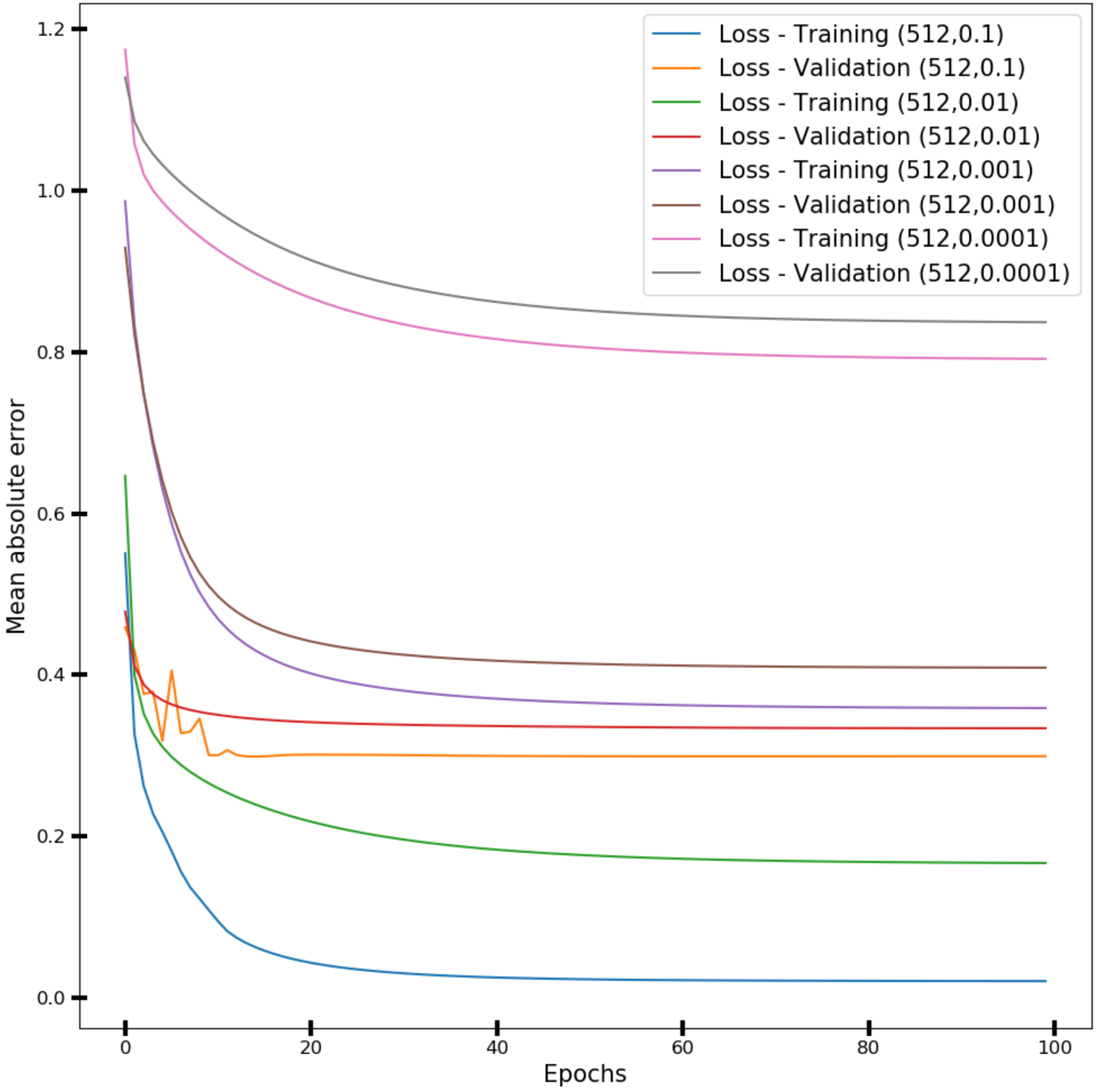


KAIST

Results : logP prediction using MLP

num_layer = 2, hidden_dim = 512

Learning rate	Test Accuracy
0.1	0.3068
0.01	0.3346
0.001	0.4043
0.0001	0.8169

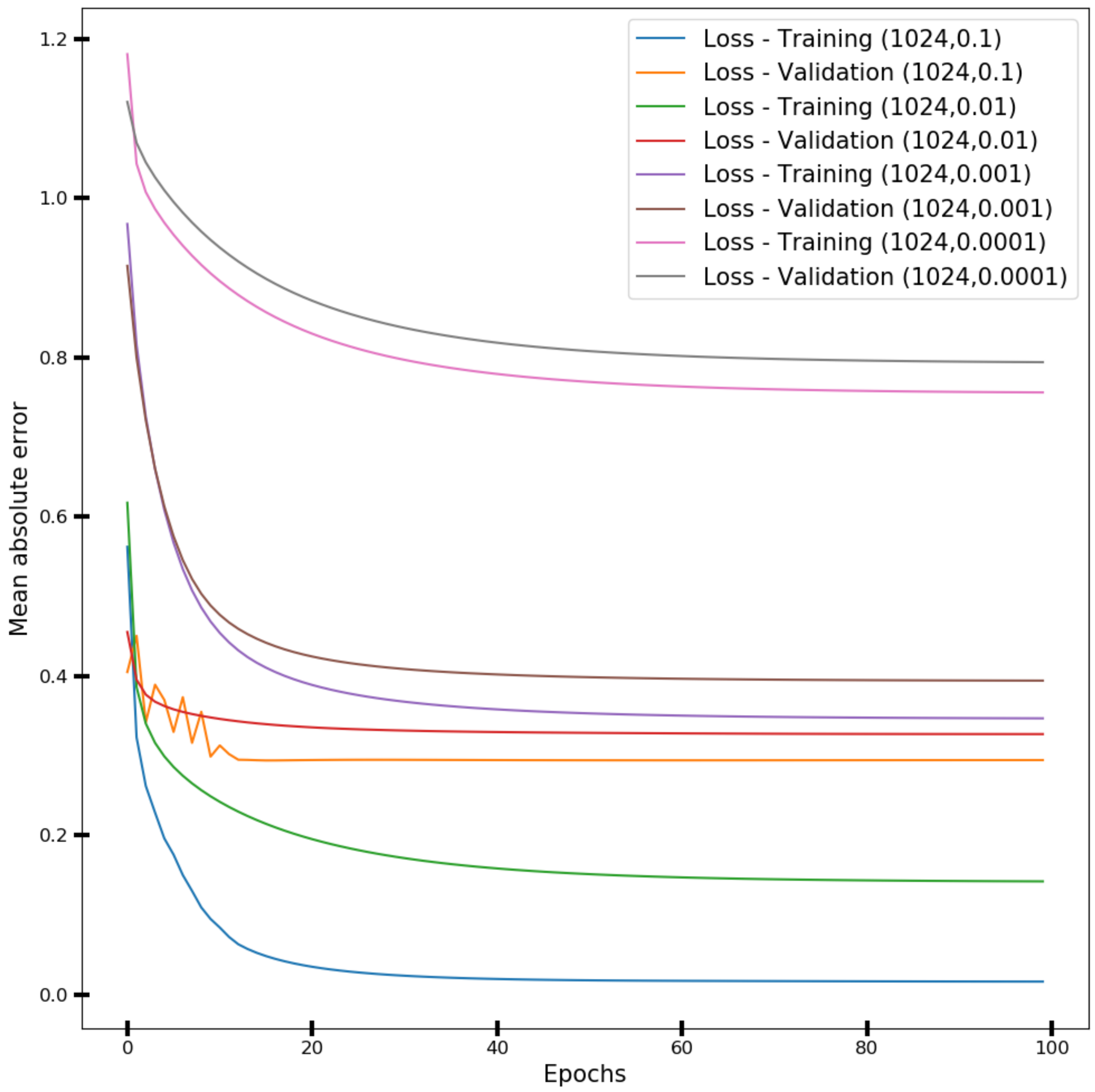


KAIST

Results : logP prediction using MLP

num_layer = 2, hidden_dim = 1024

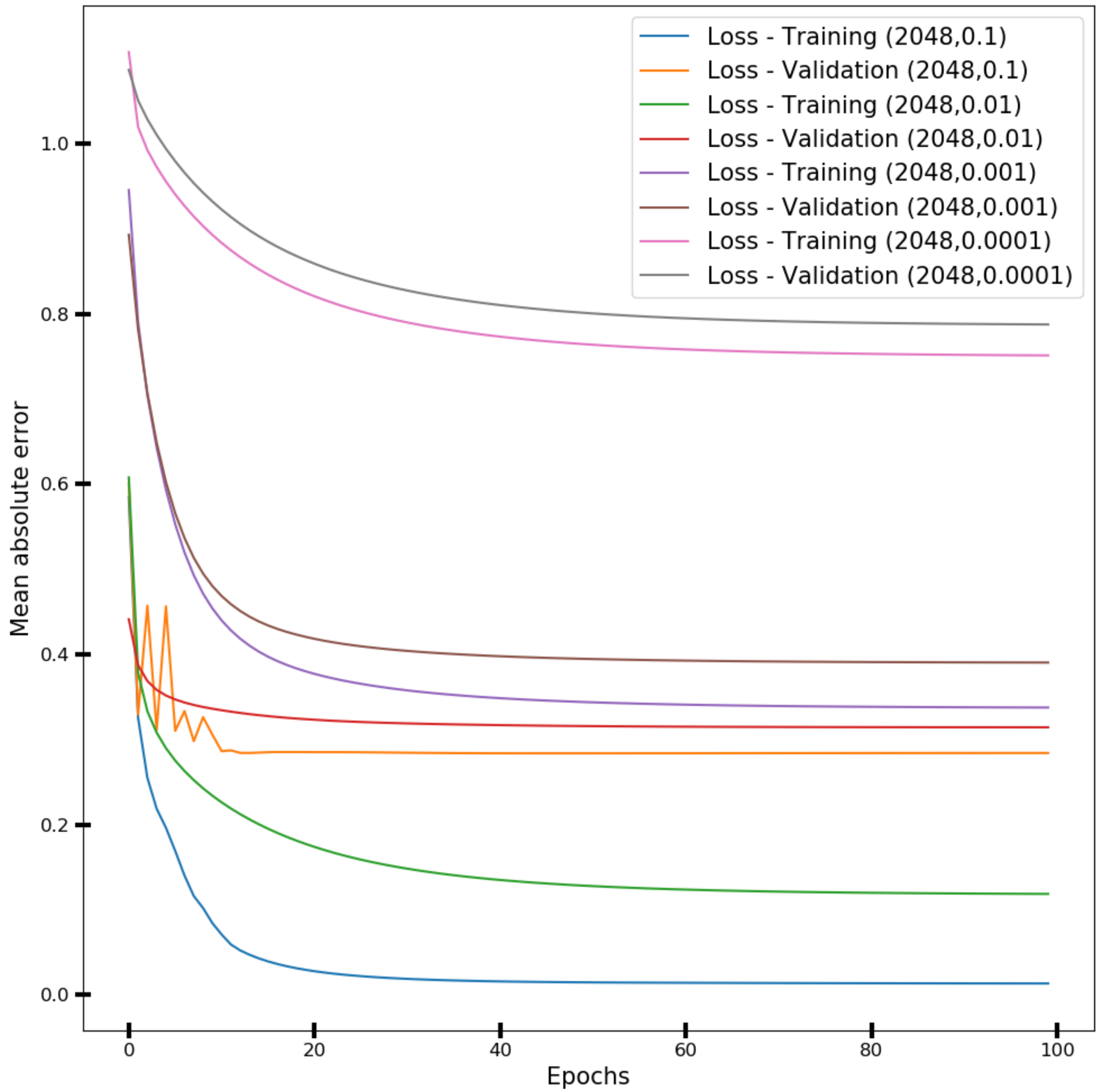
Learning rate	Test Accuracy
0.1	0.3008
0.01	0.3304
0.001	0.3929
0.0001	0.7756



Results : logP prediction using MLP

num_layer = 2, hidden_dim = 2048

Learning rate	Test Accuracy
0.1	0.2945
0.01	0.3207
0.001	0.3912
0.0001	0.7743

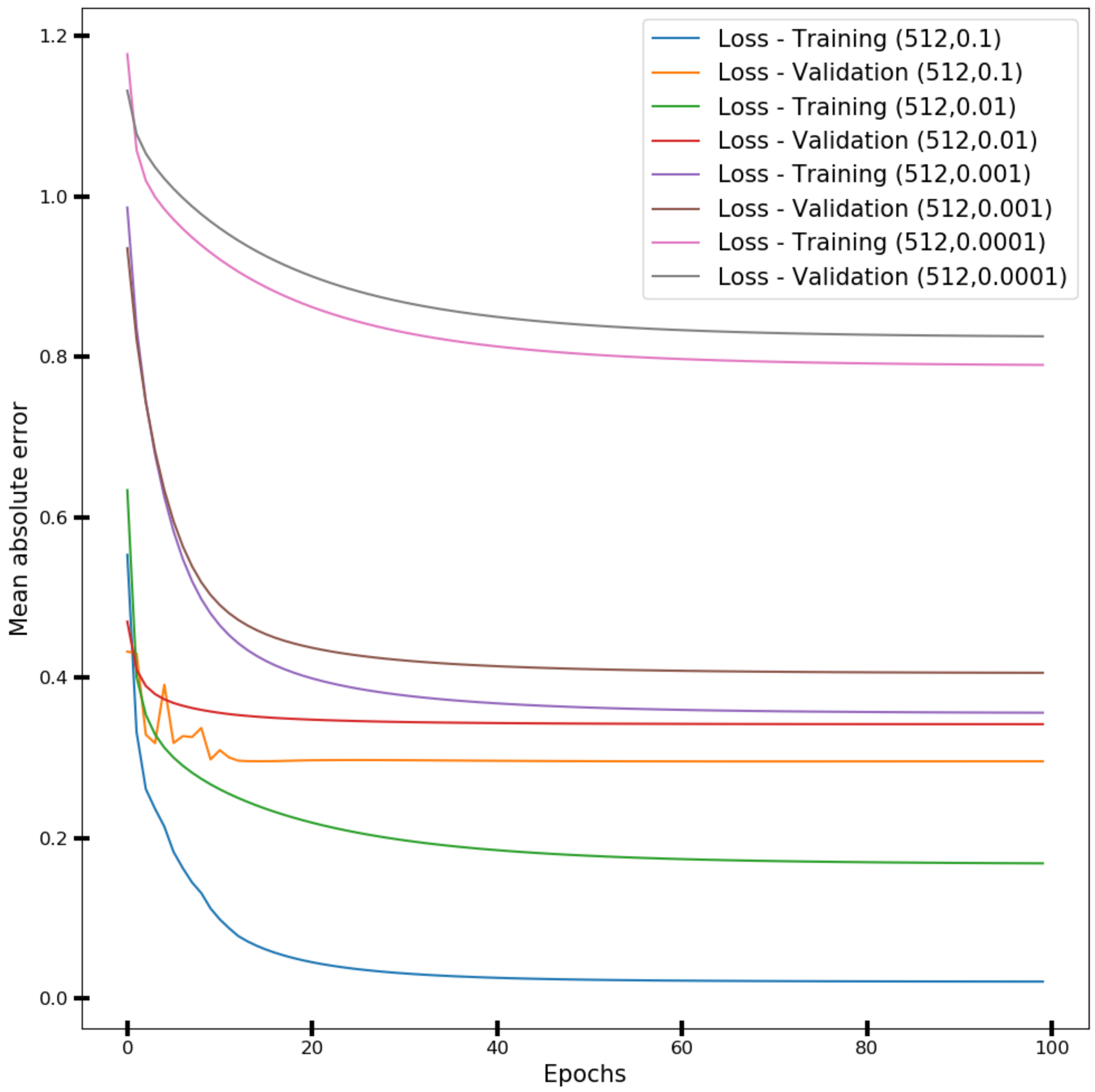


KAIST

Results : logP prediction using MLP

num_layer = 3, hidden_dim = 512

Learning rate	Test Accuracy
0.1	0.3008
0.01	0.3352
0.001	0.4024
0.0001	0.8145

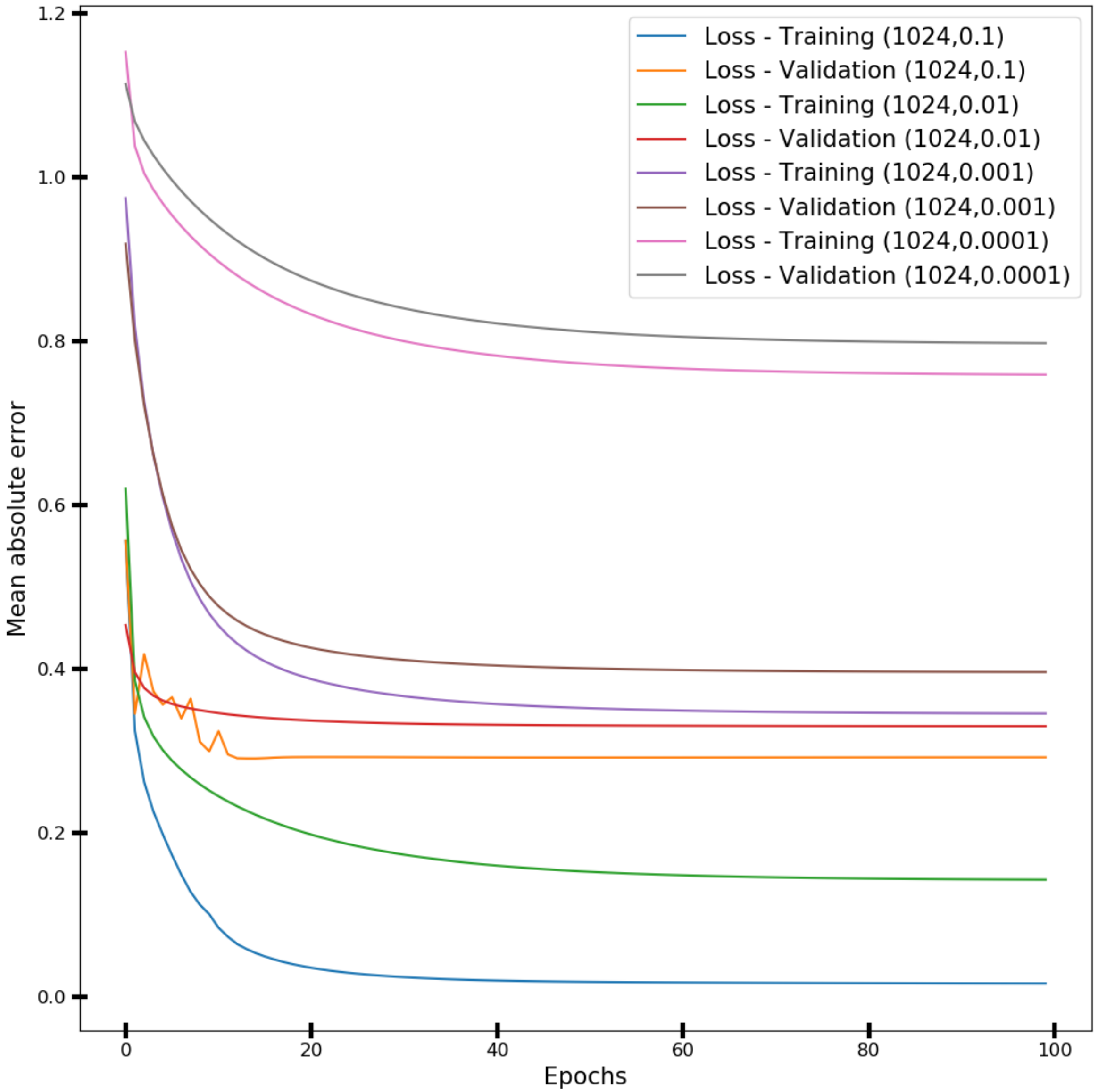


KAIST

Results : logP prediction using MLP

num_layer = 3, hidden_dim = 1024

Learning rate	Test Accuracy
0.1	0.3037
0.01	0.3323
0.001	0.3915
0.0001	0.7861

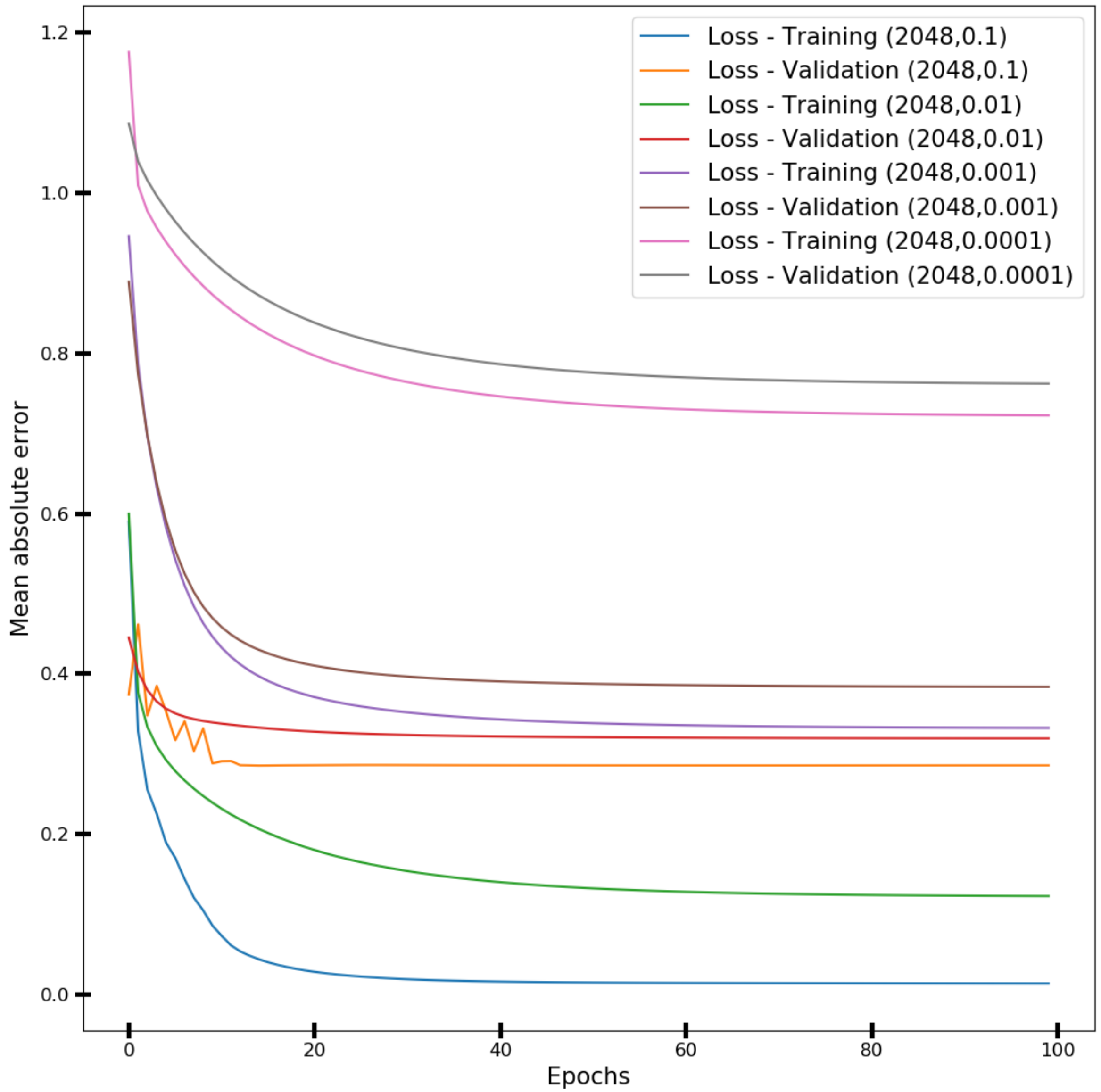


KAIST

Results : logP prediction using MLP

num_layer = 3, hidden_dim = 2048

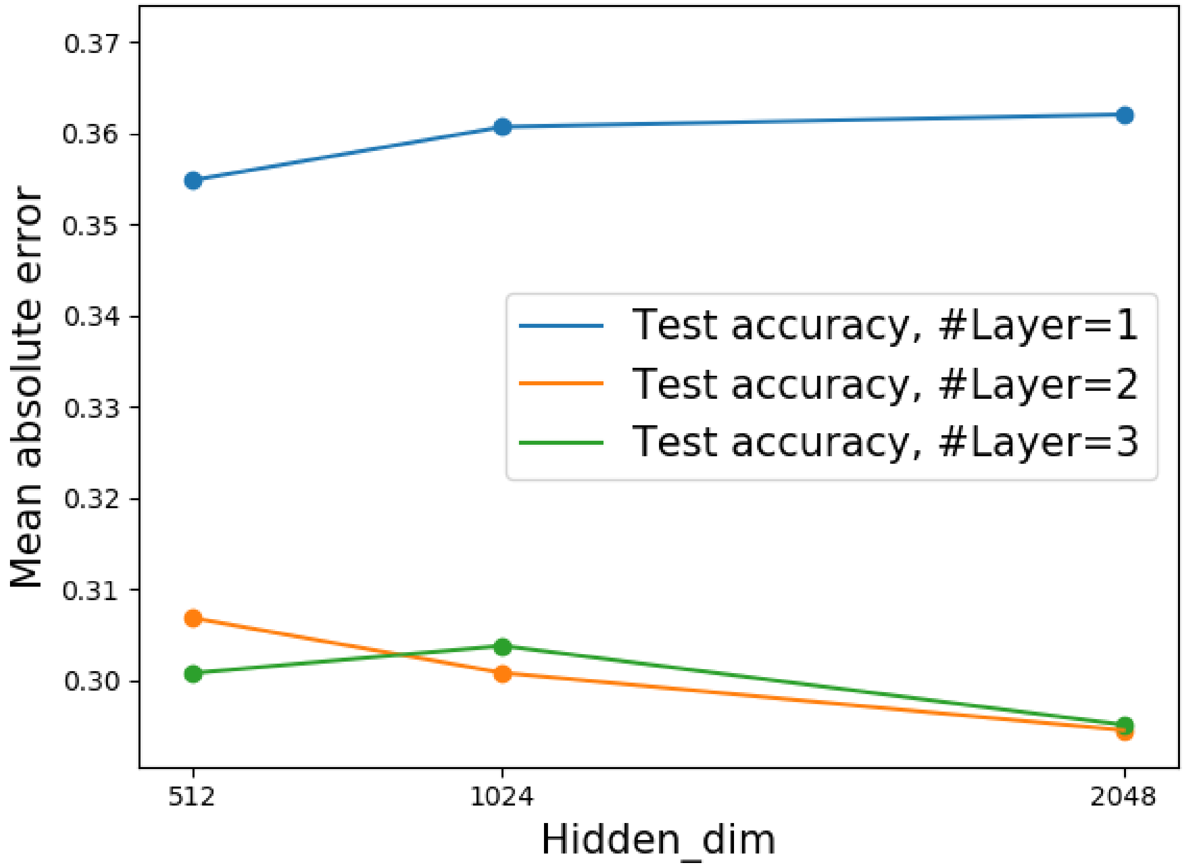
Learning rate	Test Accuracy
0.1	0.2951
0.01	0.3201
0.001	0.3821
0.0001	0.7488



KAIST

Results : logP prediction using MLP

Summary



1. Finding the best learning rate is really really important.
2. At low learning rate, it seems that a slight over-fitting might be occurred.
3. Increasing the hidden dimension can improve the test accuracy.
4. Increasing the number of layers might improve the test accuracy, however, is exposed to the over-fitting.

How can we improve the models?

Possibilities

- Learning rate is too small or big.

  → We checked this.
- Missing regularizations (prior regularization, dropout)
- Input, the molecular fingerprint, is not good.
- Need better model, instead of MLP

KAIST

How can we improve the models?

Possibilities

- Learning rate is too small or big.
- Missing regularizations (prior regularization, dropout)

  → Let's see
- Input, the molecular fingerprint, is not good.
- Need better model, instead of MLP

KAIST

Regularization

Dropout

Two options

- tf.nn.dropout(~~~)

  https://www.tensorflow.org/api_docs/python/tf/nn/dropout

- tf.layers.dropout(~~~)

  https://www.tensorflow.org/api_docs/python/tf/layers/dropout

- Difference

  https://stackoverflow.com/questions/44395547/tensorflow-whats-the-difference-between-tf-nn-dropout-and-tf-layers-dropout

Regularization

Dropout

Two options

- tf.nn.dropout(~~~)

  https://www.tensorflow.org/api_docs/python/tf/nn/dropout

```
tf.nn.dropout(
    x,
    keep_prob,
    noise_shape=None,
    seed=None,
    name=None
)
```

✓ Always turn on the dropout.

✓ keep_prob : “Probability that each element is kept”

Regularization

Dropout

Two options

- tf.layers.dropout(~~~)

  https://www.tensorflow.org/api_docs/python/tf/layers/dropout

```
tf.layers.dropout(
    inputs,
    rate=0.5,
    noise_shape=None,
    seed=None,
    training=False,
    name=None
)
```

✓ Can choose turning on or off the dropout.

✓ rate : “The dropout rate”

Regularization

Dropout

```
#2. Construct a neural network
X = tf.placeholder(tf.float64, shape=[None, 2048])
Y = tf.placeholder(tf.float64, shape=[None, ])
is_training = tf.placeholder(tf.bool, shape=())

h = X
for i in range(num_layer-1):
    h = tf.layers.dense(h,
                        units=hidden_dim,
                        use_bias=True,
                        activation=tf.nn.relu,
                        kernel_initializer=tf.contrib.layers.xavier_initializer())
    h = tf.layers.dropout(h,
                          rate=drop_rate,
                          training=is_training)
h = tf.layers.dense(h,
                    units=hidden_dim,
                    use_bias=True,
                    activation=tf.nn.tanh,
                    kernel_initializer=tf.contrib.layers.xavier_initializer())
Y_pred = tf.layers.dense(h,
                         units=1,
                         use_bias=True,
                         kernel_initializer=tf.contrib.layers.xavier_initializer())
```

Just add the dropout layer after dense layer

I do not use the dropout just before the last dense layer

KAIST

Regularization

Dropout

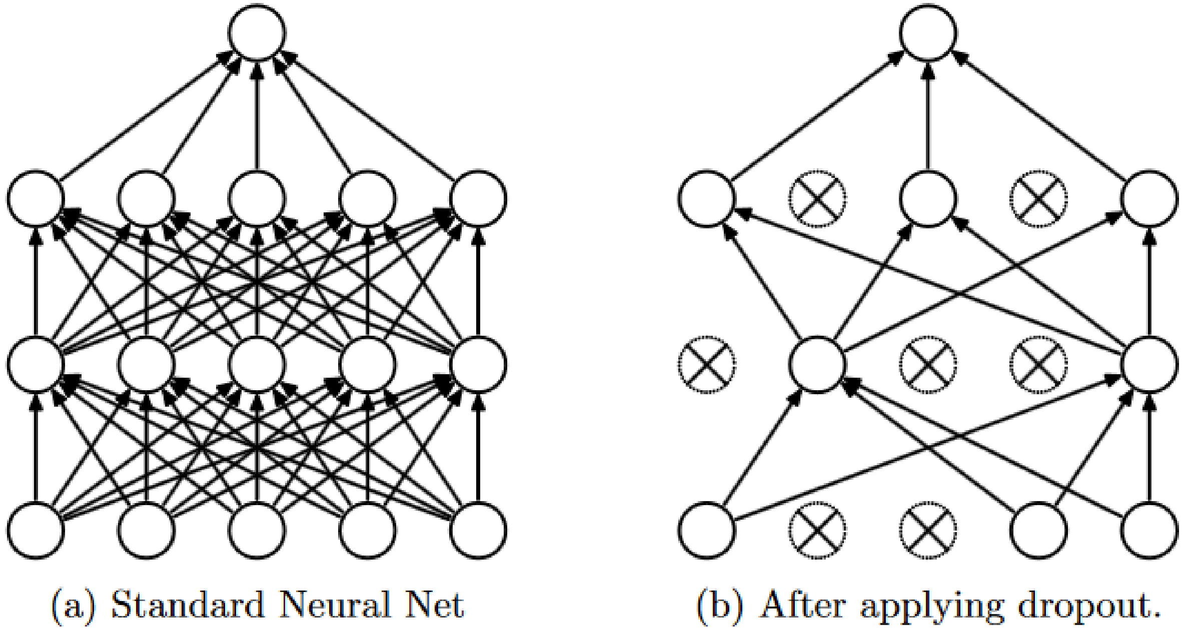


Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Srivastava, Nitish, et al.
"Dropout: a simple way to prevent neural networks from overfitting."
The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

Regularization

Dropout

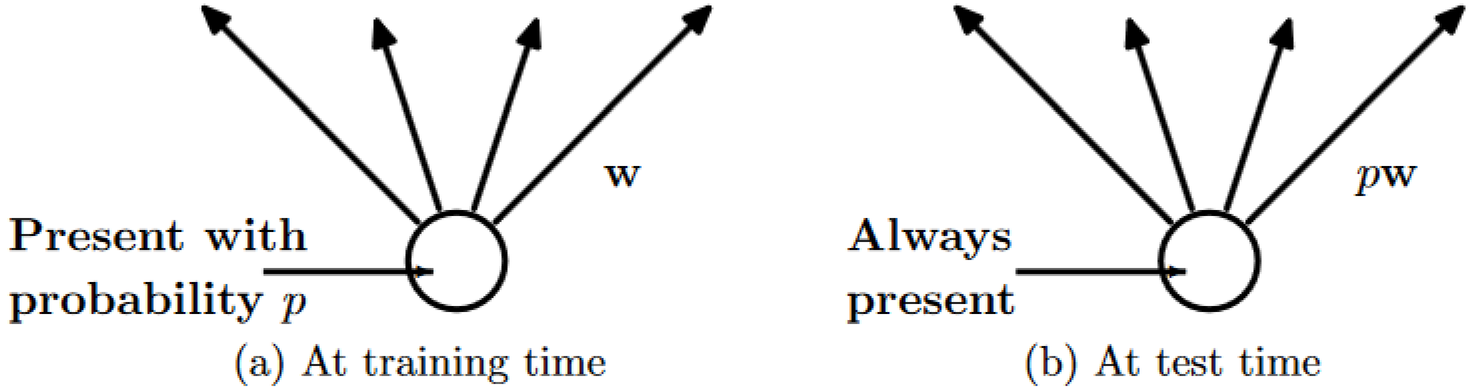


Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

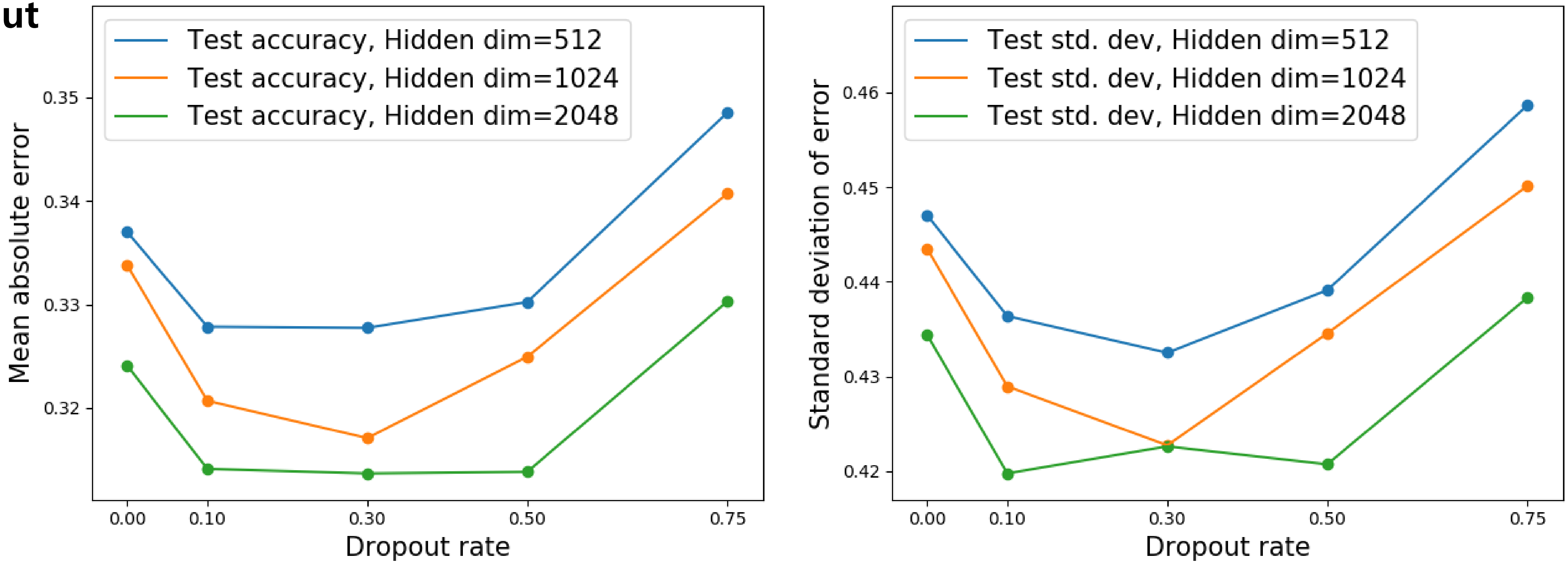
STRONGLY RECOMMEND to read this paper, which introduced the dropout for the first time

Srivastava, Nitish, et al.
"Dropout: a simple way to prevent neural networks from overfitting."
The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

KAIST

Regularization

Dropout

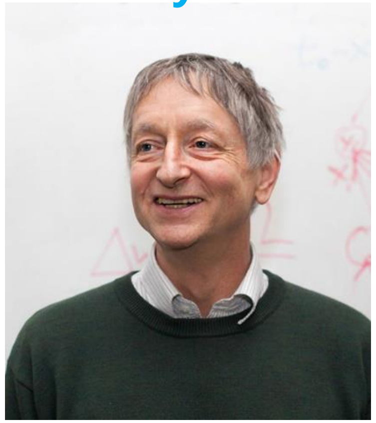


- # Hidden layers = 2
- Learning rate = 0.01

1. Using dropout can improve the vanilla MLP.
2. Question) How can we obtain the best dropout rate without manual searching?

Regularization

Geoffrey Hinton



Dropout

- Dropout : A simple way to prevent neural networks from overfitting
- Variational Dropout and the Local Reparameterization Trick
- Dropout as a Bayesian Approximation

  : Representing Model Uncertainty in Deep Learning
- Risk versus Uncertainty in Deep Learning

  : Bayes, Bootstrap and the Dangers of Dropout
- Concrete Dropout
- Pushing the bounds of dropout

Zoubin Ghahramani

Max Welling

Yarin Gal

Durk Kingma

A lot of articles introduces the concept, usages and applications of the dropout.

~~(Do not need to read them all…)~~

Regularization

Regularization

```
tf.layers.dense(
    inputs,
    units,
    activation=None,
    use_bias=True,
    kernel_initializer=None,
    bias_initializer=tf.zeros_initializer(),
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    trainable=True,
    name=None,
    reuse=None
)
```

```
h = X
for i in range(num_layer-1):
    h = tf.layers.dense(h,
                        units=hidden_dim,
                        use_bias=True,
                        activation=tf.nn.relu,
                        kernel_initializer=tf.contrib.layers.xavier_initializer(),
                        kernel_regularizer=tf.contrib.layers.l2_regularizer(scale=reg_scale),
                        bias_regularizer=tf.contrib.layers.l2_regularizer(scale=reg_scale))
h = tf.layers.dense(h,
                    units=hidden_dim,
                    use_bias=True,
                    activation=tf.nn.tanh,
                    kernel_initializer=tf.contrib.layers.xavier_initializer(),
                    kernel_regularizer=tf.contrib.layers.l2_regularizer(scale=reg_scale),
                    bias_regularizer=tf.contrib.layers.l2_regularizer(scale=reg_scale))
Y_pred = tf.layers.dense(h,
                         units=1,
                         use_bias=True,
                         kernel_initializer=tf.contrib.layers.xavier_initializer(),
                         kernel_regularizer=tf.contrib.layers.l2_regularizer(scale=reg_scale),
                         bias_regularizer=tf.contrib.layers.l2_regularizer(scale=reg_scale))
```

Regularization

Regularization

tf.contrib.layers.l2_regularizer

☆☆☆☆☆

```
tf.contrib.layers.l2_regularizer(
    scale,
    scope=None
)
```

Defined in `tensorflow/contrib/layers/python/layers/regularizers.py`.

See the guide: Layers (contrib) > Regularizers

Returns a function that can be used to apply L2 regularization to weights.

Small values of L2 can help prevent overfitting the training data.

Args:

- `scale` : A scalar multiplier `Tensor` . 0.0 disables the regularizer.
- `scope` : An optional scope name.

Returns:

A function with signature `l2(weights)` that applies L2 regularization.

https://www.tensorflow.org/api_docs/python/tf/contrib/layers/l2_regularizer

Regularization

Regularization

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

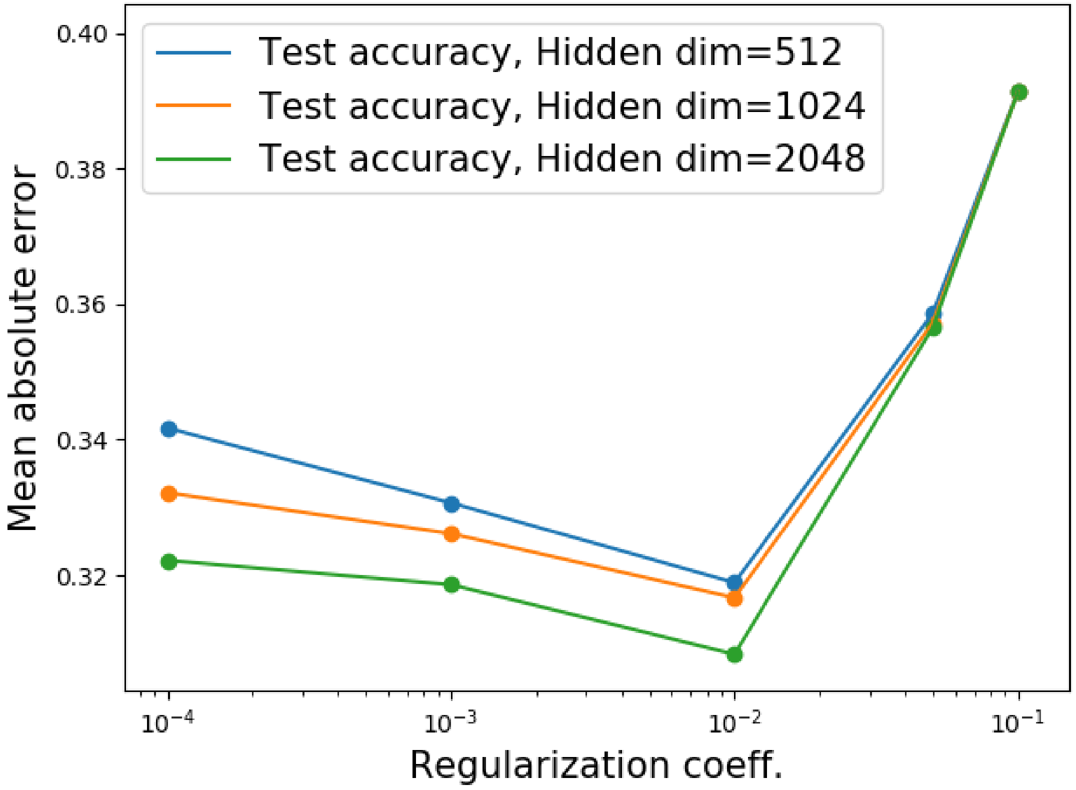
L2 Regularization

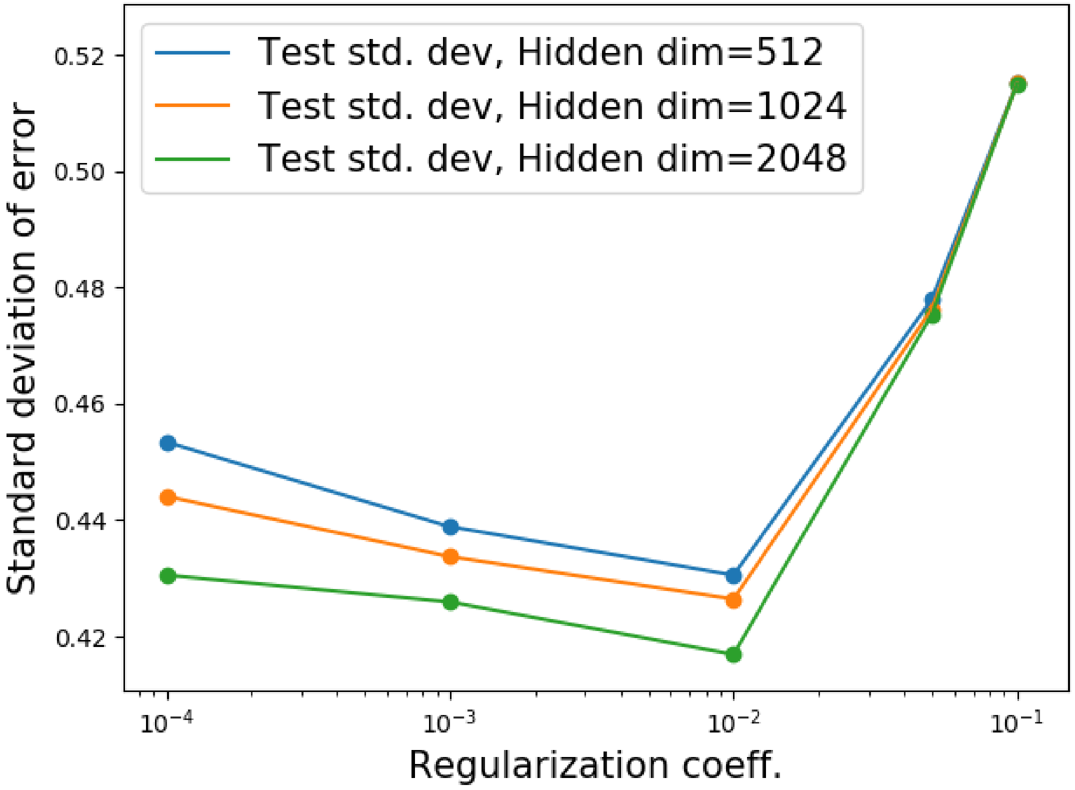
$$\text{Cost} = \underbrace{\sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

```
#3. Set a loss function, in this case we will use a MSE-loss (l2-norm)
Y_pred = tf.reshape(Y_pred, shape=[-1,])
reg_loss = tf.losses.get_regularization_loss()
loss = tf.reduce_mean( (Y_pred - Y)**2 ) + reg_loss
```

Regularization

Regularization



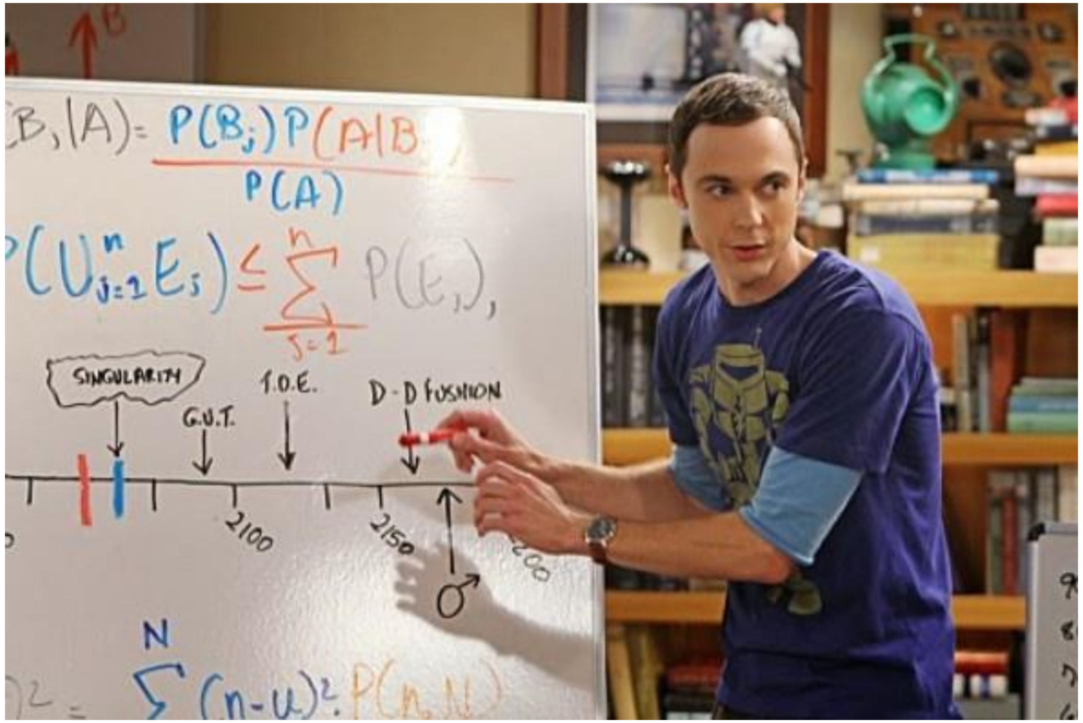


- # Hidden layers = 2
- Learning rate = 0.01

Using L2-regularization can improve the vanilla MLP.

Regularization

Regularization



Regularization can be interpreted in **Bayesian's view**.

See below articles.

https://medium.com/autonomous-agents/bayesian-regularization-for-neuralnetworks-2f2d34f03adc

https://en.wikipedia.org/wiki/Bayesian_interpretation_of_kernel_regularization

How can we improve the models?

Possibilities

- Learning rate is too small or big.
- Missing regularizations (prior regularization, dropout)
- Input, the molecular fingerprint, is not good.
- Need better model, instead of MLP

→ Using raw input, e.g.) SMILES, molecular graph

rather than featurized inputs, e.g.) molecular fingerprint

→ Using better model, e.g.) CNN, RNN, Graph NN.

KAIST

Assignment #3

Toxicity prediction model using MLP

- We constructed a model which classify molecules are toxic or not using support vector machine.
- In this assignment, train the model with MLP
- You can reuse the script, which loads and splits the data used in assignment #2.
- Report your works
  - without using dropout and regularization
  - with using dropout and regularization
  - compare the performance to that of the SVM