Practice 03

Implementation of MLP with TensorFlow

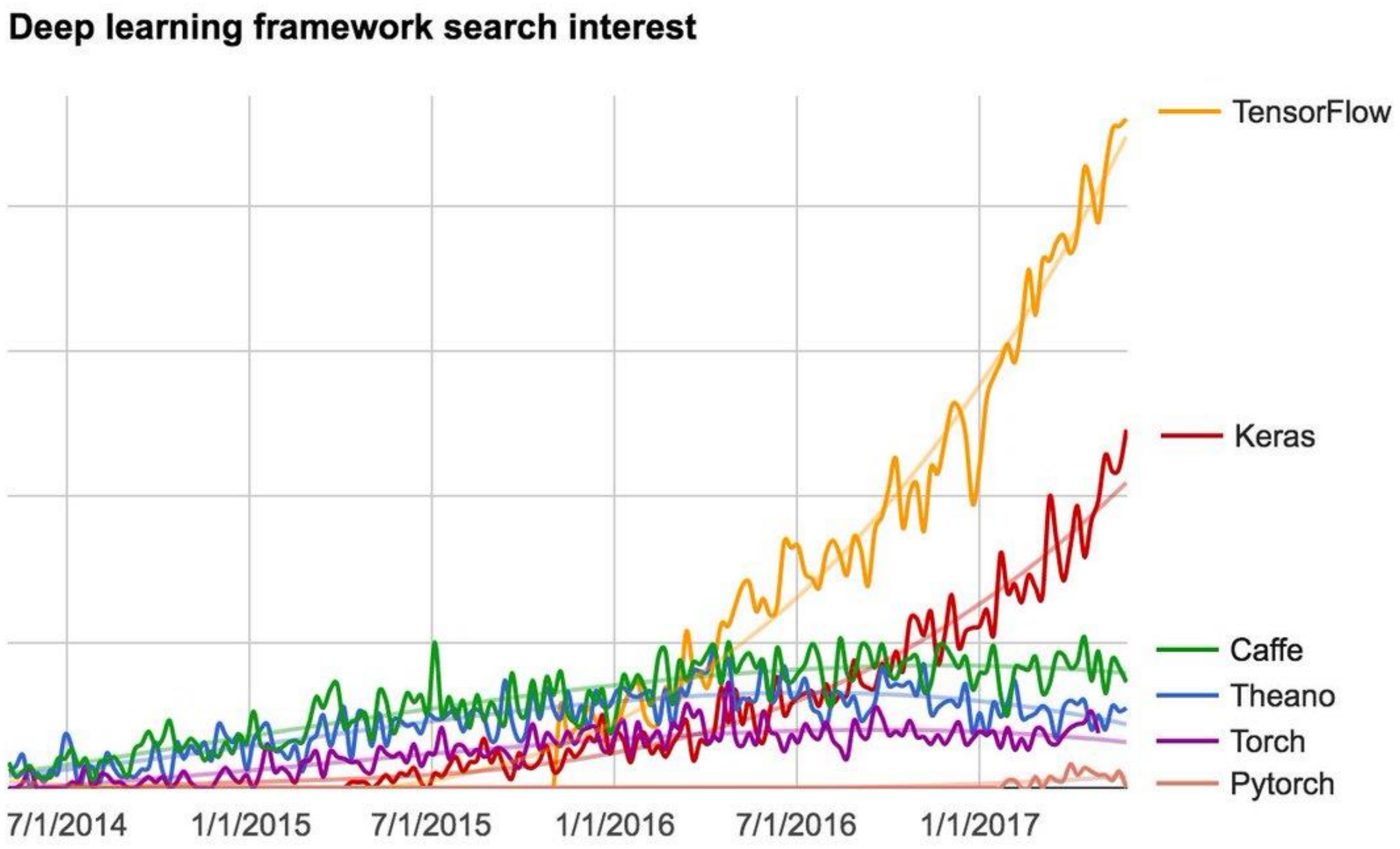
Seongok Ryu

Department of Chemistry, KAIST

Contents

- TensorFlow
- Multi-layer perceptron (MLP)
- Prediction of logP using MLP

Deep learning Frameworks - TensorFlow

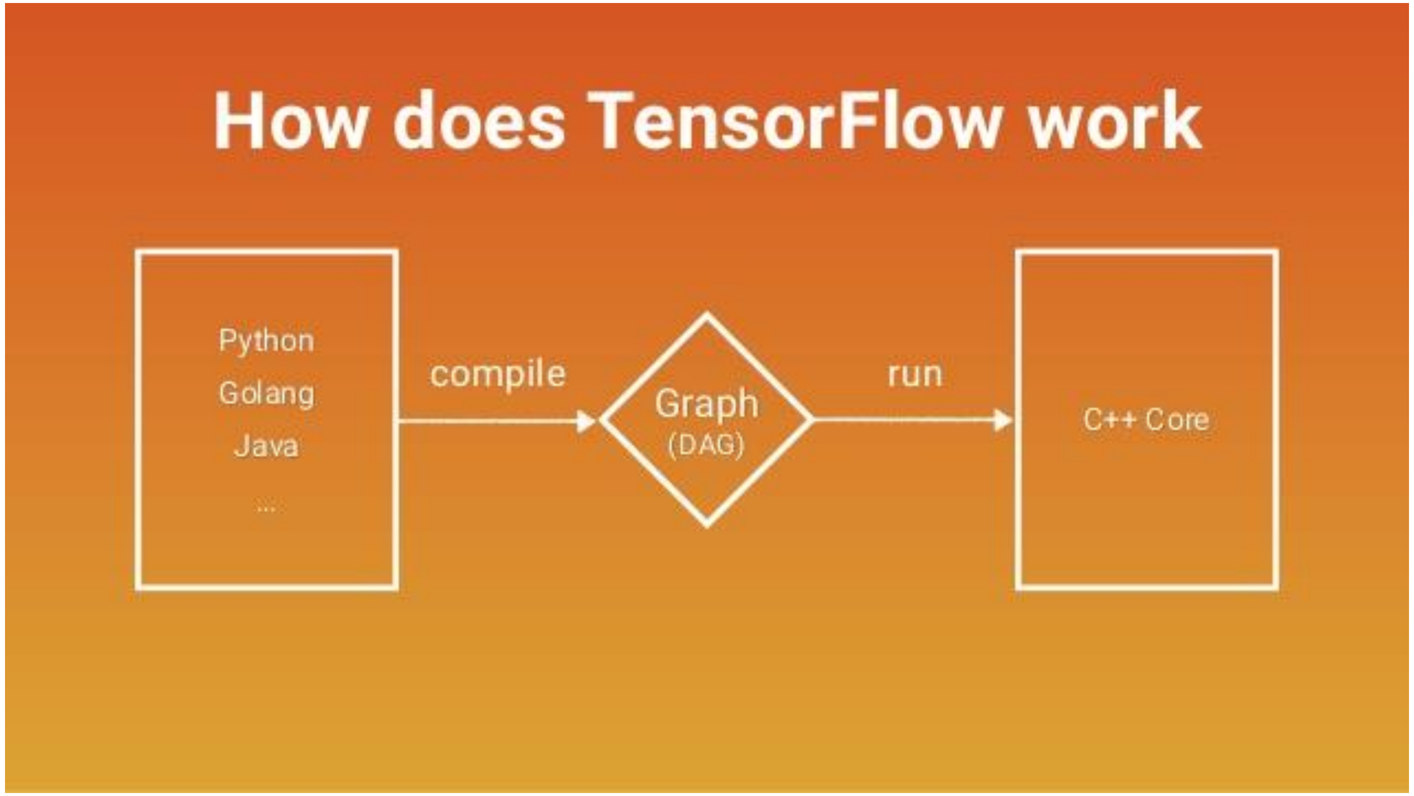


https://twitter.com/fchollet/status/871089784898310144

Deep learning Frameworks - TensorFlow

- **TensorFlow – We will use this in this class.**
  - Developed and maintained by Google
  - Most popular deep learning framework → many examples on gitHub
  - Static computational graph, *but dynamic computational graph is also supported now.*
- **Keras**
  - Framework using tensorflow and theanoas back-end
  - Super super easy to use
  - Static computational graph
- **PyTorch**
  - Developed and maintained by Facebook
  - Dynamic Computational Graph

TensorFlow



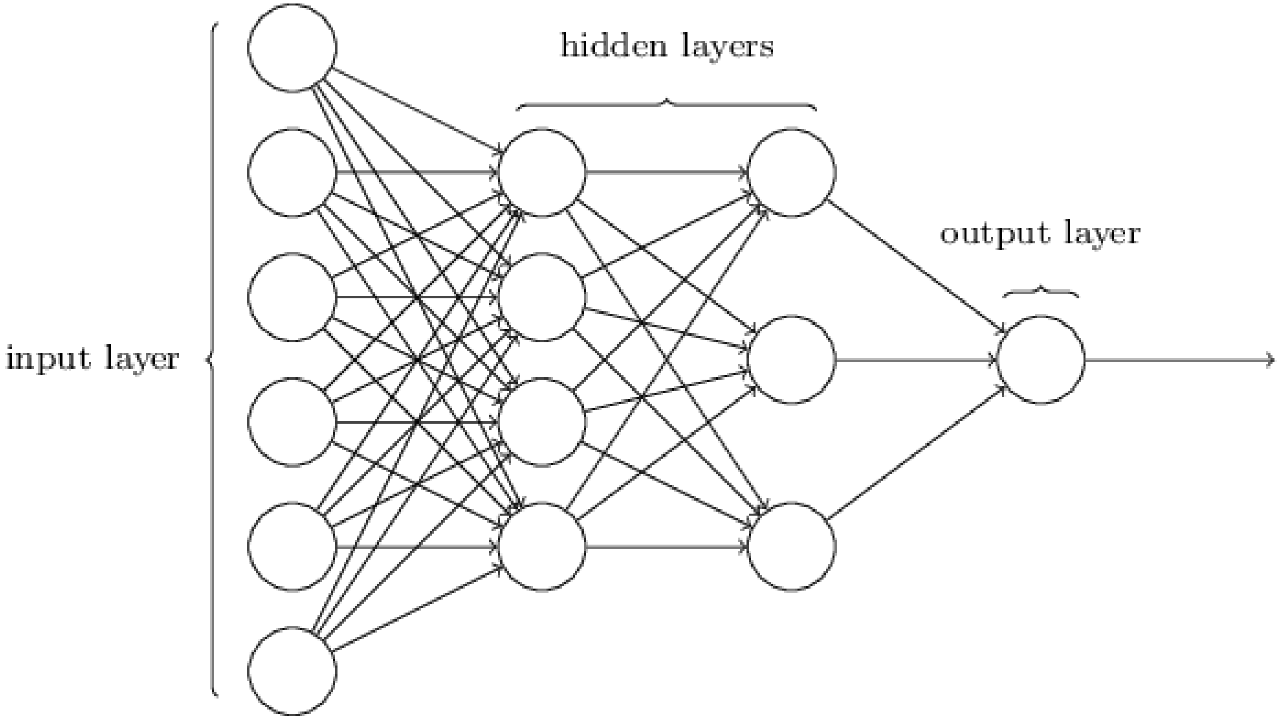
https://www.slideshare.net/jiangjun1990/gdgshanghai-2017-tensorflow-summit-recap

TensorFlow

Overall procedure

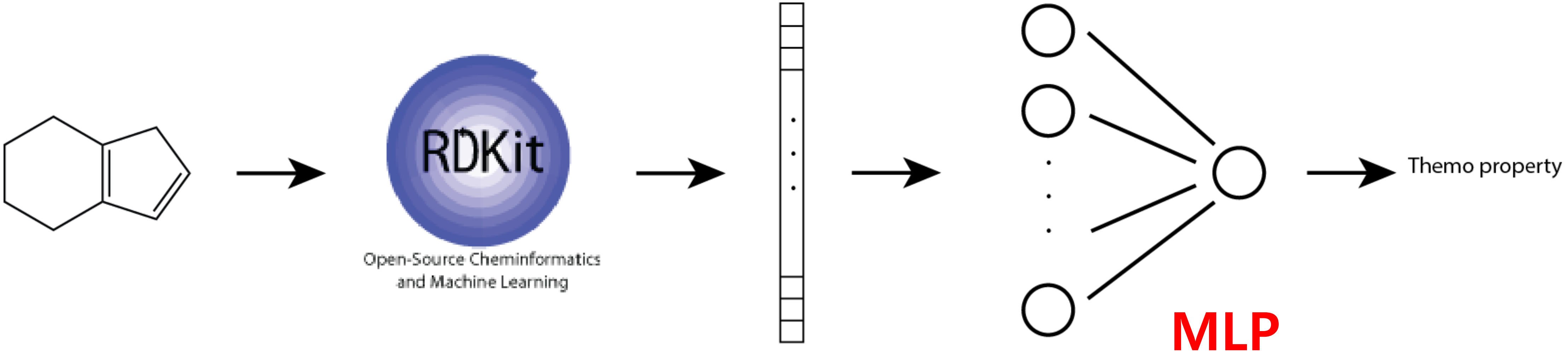
1. Prepare data
2. Construct a neural network (computational graph)
3. Set a loss function
4. Set an optimizer
5. Training & validation
6. Test

Multi-layer perceptron



$Y = \boldsymbol{f}(X)$, Using **MLP** for a function approximator

Prediction of logP using MLP

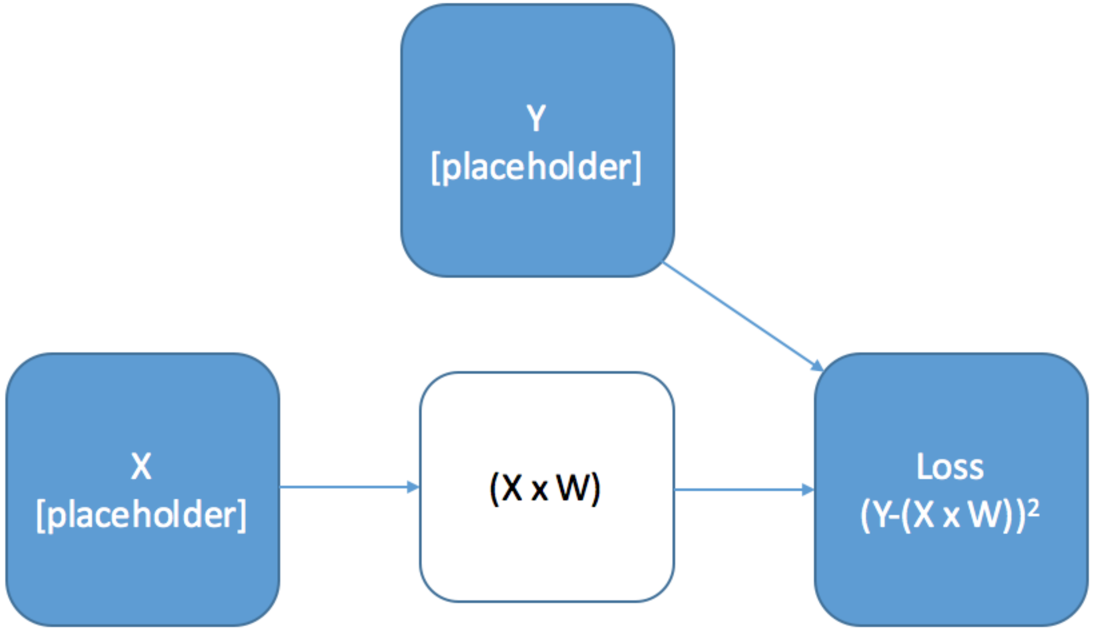


$Y = \boldsymbol{f}(X)$, Using **MLP** for a function approximator

- Y : molecular property (logP)
- X : molecular structure (fingerprint)

Multi-layer perceptron

Placeholder



- X = tf.placeholder(tf.float64, shape = [None, 2048])
- Y = tf.placeholder(tf.float64, shape = [None,])

Multi-layer perceptron

Construct a neural network

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

```
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,
    trainable=True,
    name=None,
    reuse=None
)
```

- **h1** = tf.layers.dense(**X**, **units=512**, activation=**tf.nn.relu**, **use_bias = True**) $h_1 = ReLU(W_1 X + b_1)$
- **h2** = tf.layers.dense(**h1**, **units=512**, activation=**tanh** , **use_bias = True**) $h_2 = tanh(W_2 h_1 + b_2)$
- **output** = tf.layers.dense(**h2**, **units=1**, activation=**None** , **use_bias = True**) $output = tanh(W_2 h_2 + b_2)$

Multi-layer perceptron

Construct a neural network

- **h1** = tf.layers.dense(**X**, **units=512**, activation=**tf.nn.relu**, **use_bias = True**)
- **h2** = tf.layers.dense(**h1**, **units=512**, activation=**tanh** , **use_bias = True**)
- **output** = tf.layers.dense(**h2**, **units=1**, activation=**None** , **use_bias = True**)

$$\boldsymbol{h_1} = \boldsymbol{ReLU}(\boldsymbol{W_1 X} + \boldsymbol{b_1})$$

$$\boldsymbol{h_2} = \boldsymbol{tanh}(\boldsymbol{W_2 h_1} + \boldsymbol{b_2})$$

$$\boldsymbol{output} = \boldsymbol{tanh}(\boldsymbol{W_2 h_2} + \boldsymbol{b_2})$$

Question) Dimension of $\boldsymbol{W_1, W_2, W_3, b_1, b_2, b_3, h_1, h_2, output}$?

KAIST

Multi-layer perceptron

Construct a neural network

- h1 = tf.layers.dense(X, units=512, activation=tf.nn.relu, use_bias = True)
- h2 = tf.layers.dense(h1, units=512, activation=tanh , use_bias = True)
- output = tf.layers.dense(h2, units=1, activation=None , use_bias = True)

$$h_1 = ReLU(W_1X + b_1)$$

$$h_2 = tanh(W_2h_1 + b_2)$$

$$output = tanh(W_2h_2 + b_2)$$

Question) Dimension(shape) of $W_1, W_2, W_3, b_1, b_2, b_3, h_1, h_2, output$?

Answer)

W_1: [2048, 512], W_2 : [512, 512], W_3 : [512, 1]

b_1: [512], b_2 : [512], b_3 : [1]

h_1: [None, 512], h_2 : [None, 512], $output$: [None, 1]

KAIST

Multi-layer perceptron

Set a loss function

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\left(y_{i,truth} - y_{i,pred}\right)^2$$

output = tf.flatten(output) # Change shape of tensor from **[None, 1]** to **[None,]**

loss = tf.reduce_mean((Y – output)**2)

Set an optimizer

lr = tf.Variable(0.0, trainable=False) # learning rate

loss = tf.reduce_mean((Y – output)**2)

opt = tf.train.AdamOptimizer(lr).minimize(loss)

sess =tf.Session()

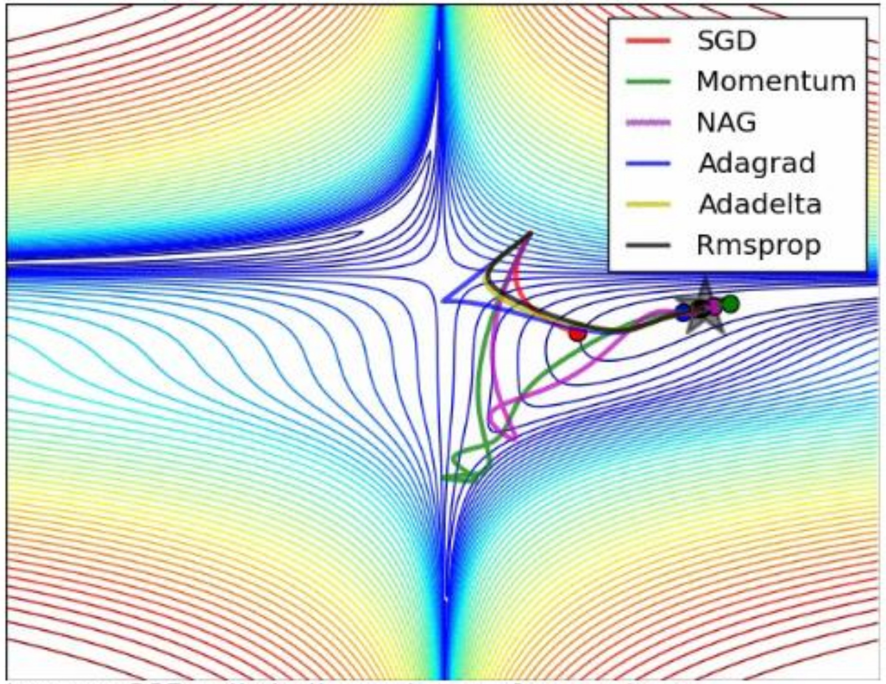
Init = tf.global_variables_initializer()

sess.run(init)

Multi-layer perceptron

Set an optimizer

: many different optimizers exist



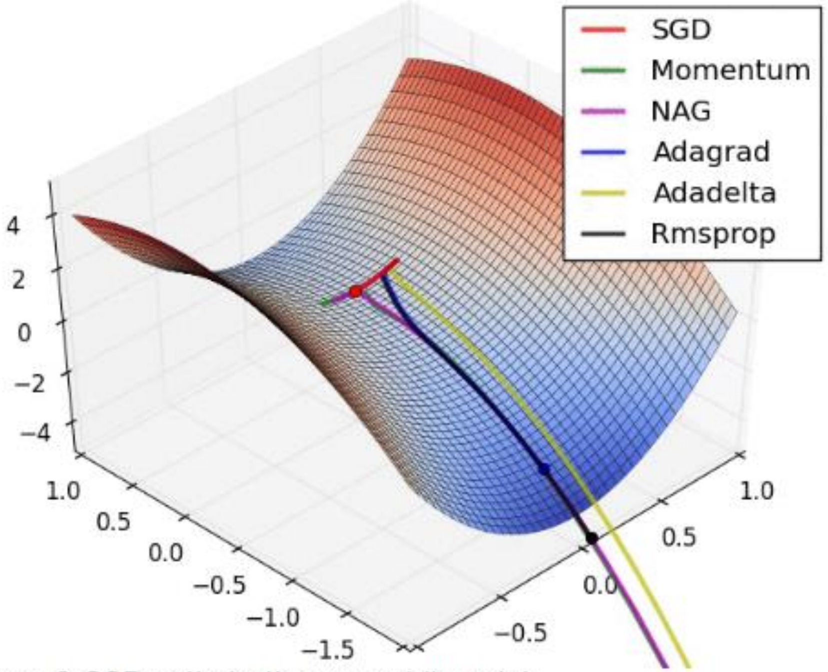


Image 5: SGD optimization on loss surface contours

Image 6: SGD optimization on saddle point

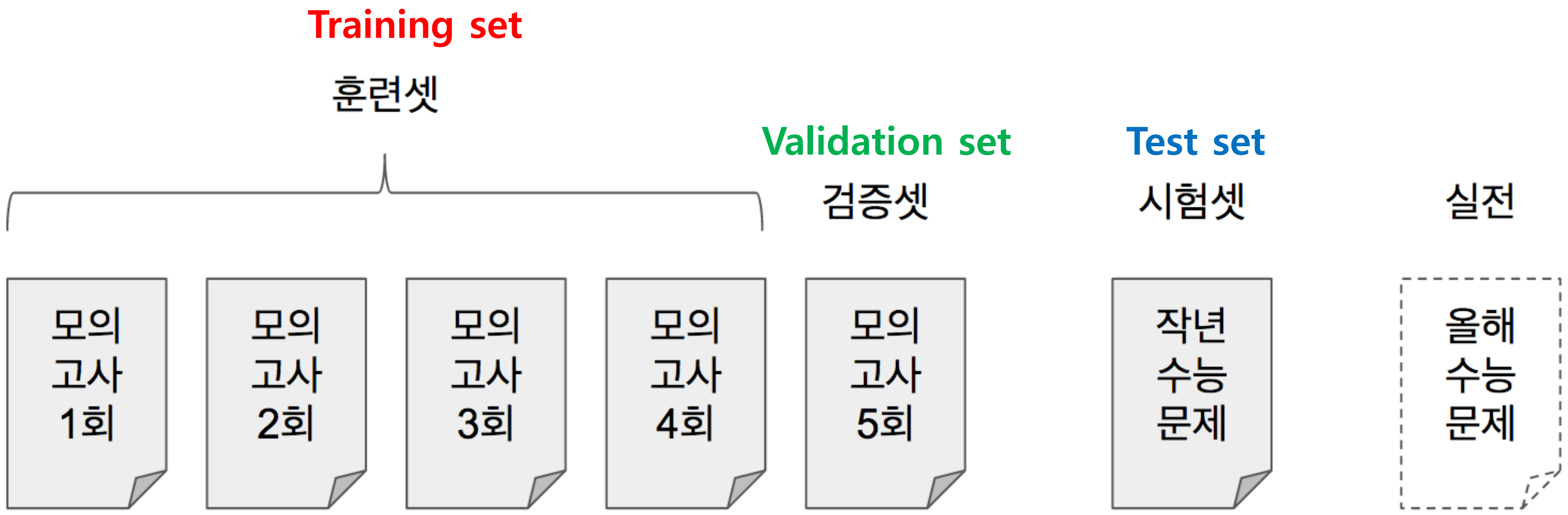
As we can see, the adaptive learning-rate methods, i.e. Adagrad, Adadelta, RMSprop, and Adam are most suitable and provide the best convergence for these scenarios.

http://ruder.io/optimizing-gradient-descent/

$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\left(y_{i,truth} - y_{i,pred}\right)^2$, in our study

Multi-layer perceptron

Training & validation



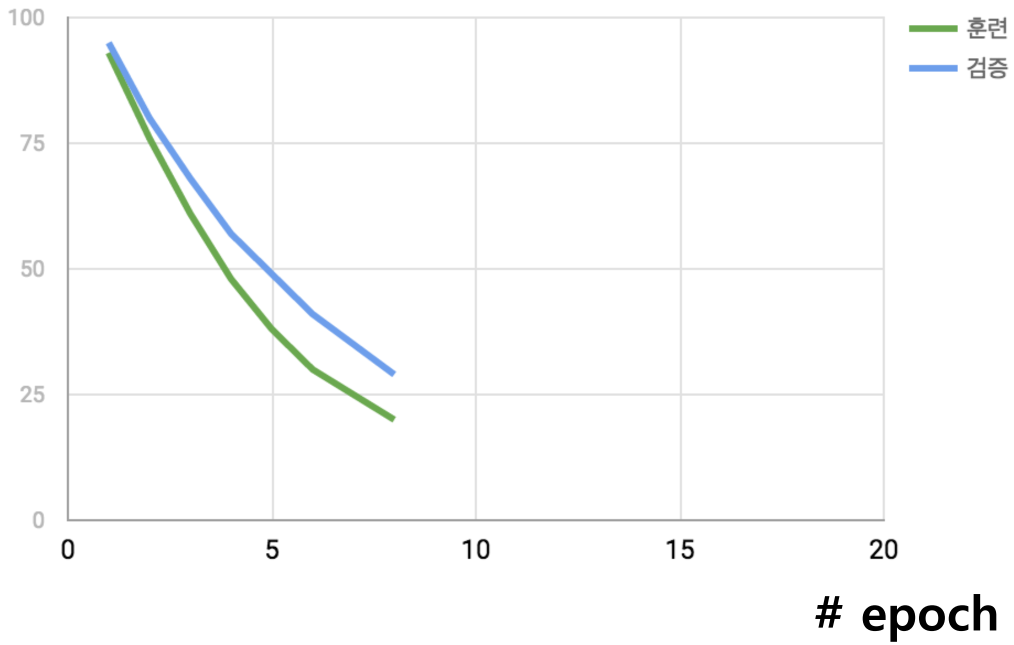
- Use the **training set** for train the neural network
- Use the **validation set** for check whether the neural network is successfully being trained
- Use the **test set** for check the performance of trained neural network.

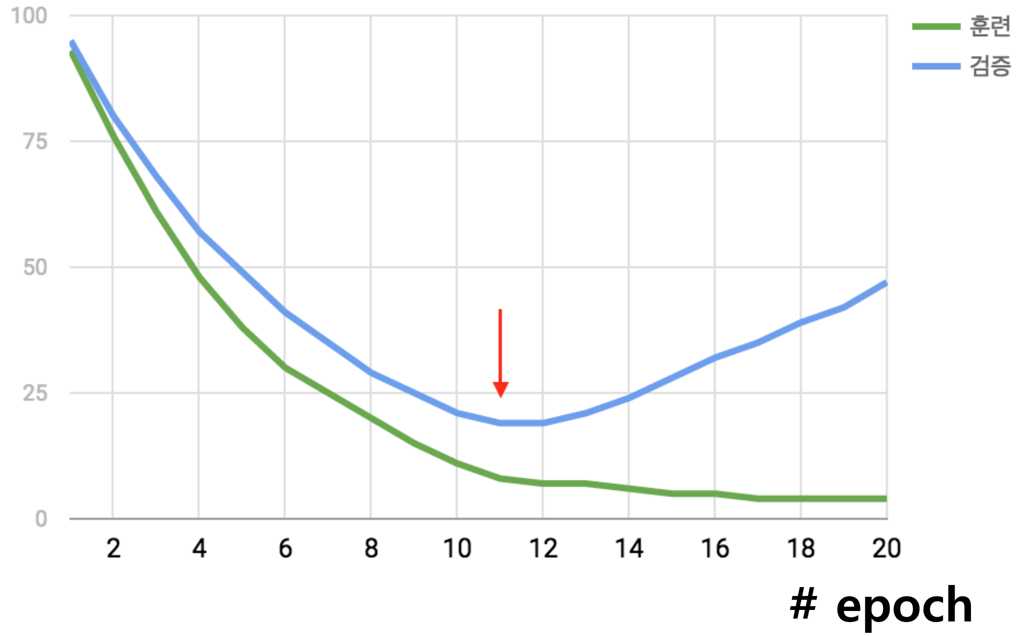
KAIST

https://tykimos.github.io/2017/03/25/Dataset_and_Fit_Talk/

Multi-layer perceptron

Training & validation





- Use the **training set** for train the neural network
- Use the **validation set** for check whether the neural network is successfully being trained
- Use the **test set** for check the performance of trained neural network.

https://tykimos.github.io/2017/03/25/Dataset_and_Fit_Talk/

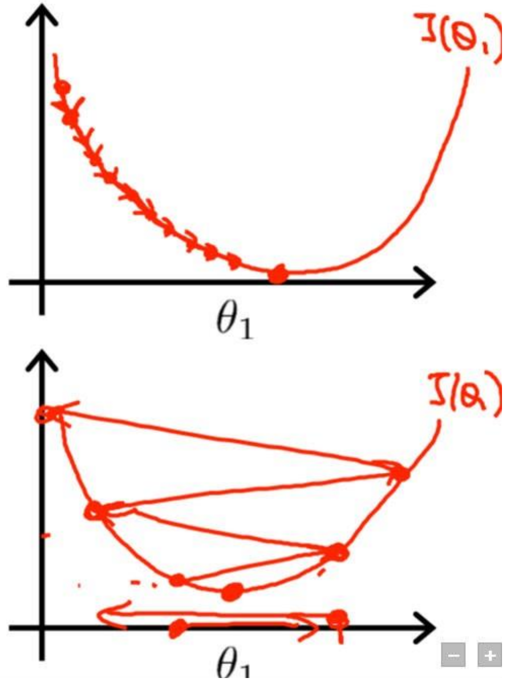
Multi-layer perceptron

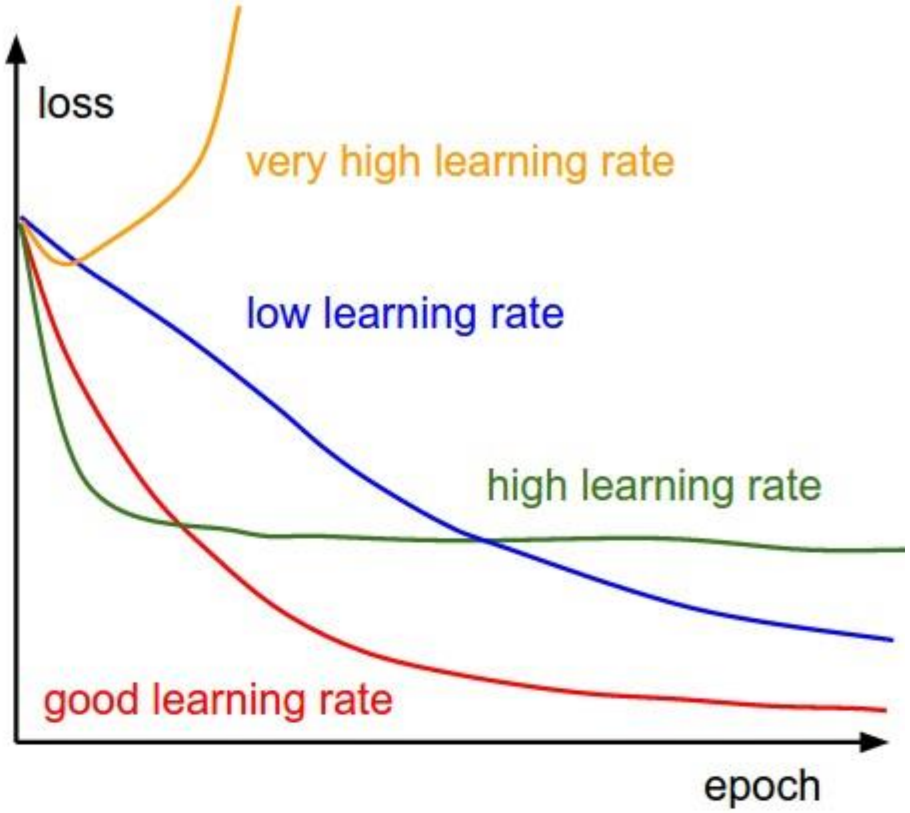
Learning rate

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.

If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



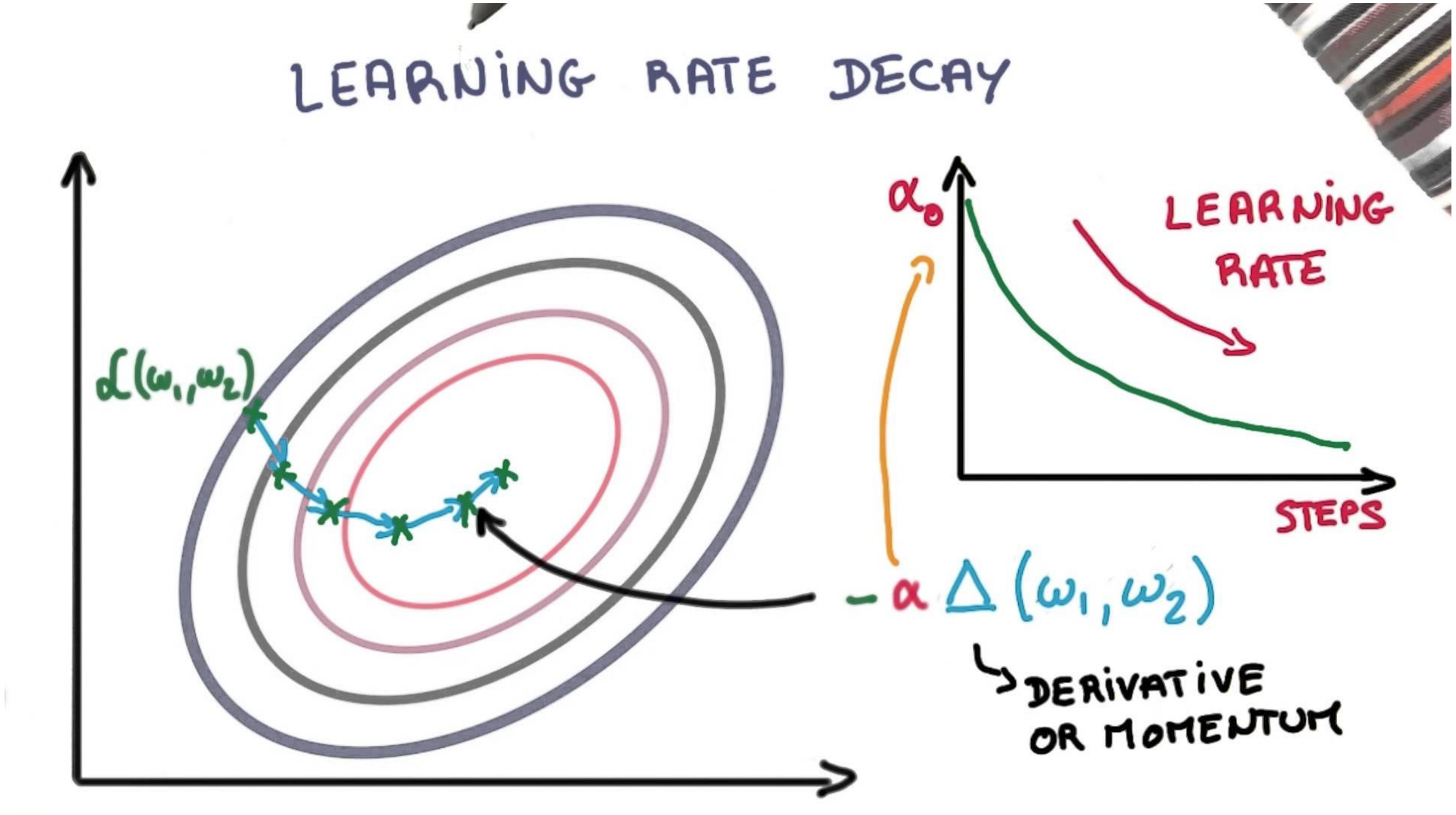


https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0

http://cs231n.github.io/neural-networks-3/

Multi-layer perceptron

Learning rate & decay rate



https://www.youtube.com/watch?v=s6jC7Wc9iMI

Multi-layer perceptron

Training & validation

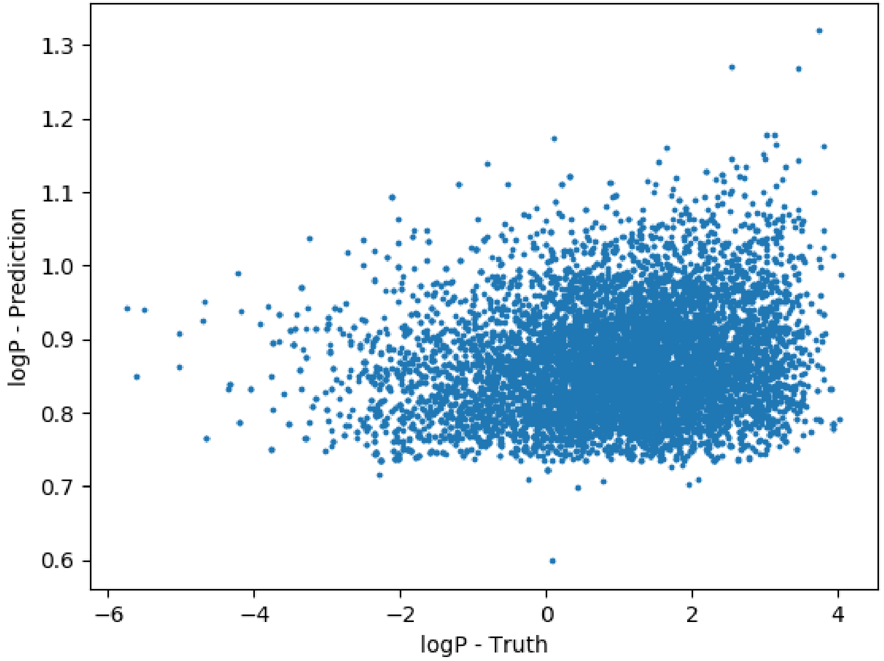
```
init_lr = 0.001
for t in range(epoch_size):

    pred_train = []
    sess.run(tf.assign( lr, init_lr*( decay_rate**t ) ))
    for i in range(batch_train):
        X_batch = fps_train[i*batch_size:(i+1)*batch_size]
        Y_batch = logP_train[i*batch_size:(i+1)*batch_size]
        _opt, _Y, _loss = sess.run([opt, Y_pred, loss], feed_dict = {X : X_batch, Y : Y_batch})
        pred_train.append(_Y.flatten())
        #print("Epoch :", t, "\t batch:", i, "Loss :", _loss, "\t Training")
    pred_train = np.concatenate(pred_train, axis=0)
    error = (logP_train-pred_train)
    mae = np.mean(np.abs(error))
    rmse = np.sqrt(np.mean(error**2))
    stdv = np.std(error)

    print ("MSE :", mae, "RMSE :", rmse, "Std :", stdv, "\t Training, \t Epoch :", t)
```

```
    pred_validation = []
    for i in range(batch_validation):
        X_batch = fps_validation[i*batch_size:(i+1)*batch_size]
        Y_batch = logP_validation[i*batch_size:(i+1)*batch_size]
        _Y, _loss = sess.run([Y_pred, loss], feed_dict = {X : X_batch, Y : Y_batch})
        #print("Epoch :", t, "\t batch:", i, "Loss :", _loss, "\t validation")
        pred_validation.append(_Y.flatten())

    pred_validation = np.concatenate(pred_validation, axis=0)
    error = (logP_validation-pred_validation)
    mae = np.mean(np.abs(error))
    rmse = np.sqrt(np.mean(error**2))
    stdv = np.std(error)

    print ("MSE :", mae, "RMSE :", rmse, "Std :", stdv, "\t Validation, \t Epoch :", t)
```

- sess.run([**opt**, Y_pred, loss], feed_dict = {X : X_batch, Y : Y_bach} - **Training**
- sess.run([Y_pred, loss], feed_dict = {X : X_batch, Y : Y_bach} - **Test**

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?

https://github.com/SeongokRyu/CH485---Artificial-Intelligence-and-Chemistry/tree/master/Practice%2003

Multi-layer perceptron

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

We will learn those in the next lecture.