Linear Regression

Kyungjae Lee, Yunho Choi, Timothy Ha
RLLAB

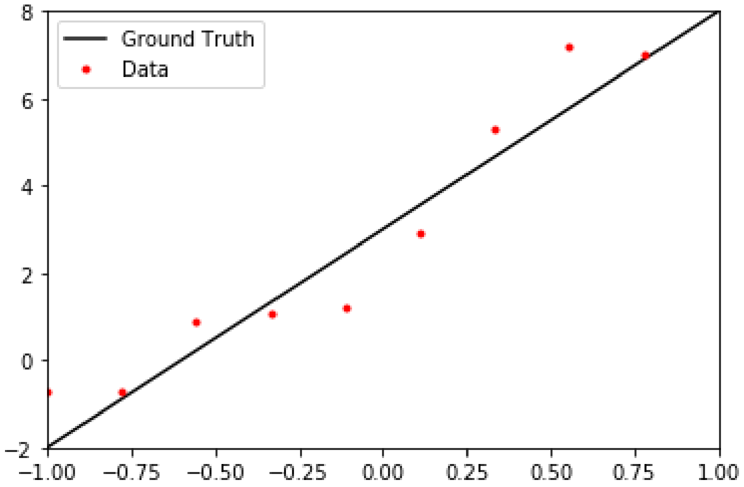
[Exercise 1] Univariate Linear Regression

- Univariate linear function: $y = w_1 x + w_0$, where $w_0, w_1 \in \mathbb{R}$.
- Let $\mathbf{w} = [w_0, w_1]^T$ and define

$$h_{\mathbf{w}}(x) = w_1 x + w_0.$$

- **Linear regression**: Given a training set $\{(x_i, y_i) : 1 \le i \le N\}$, find $h_{\mathbf{w}}$ that best fits the training set.

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N} L_2(y_j, h_{\mathbf{w}}(x_j)) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2.$$

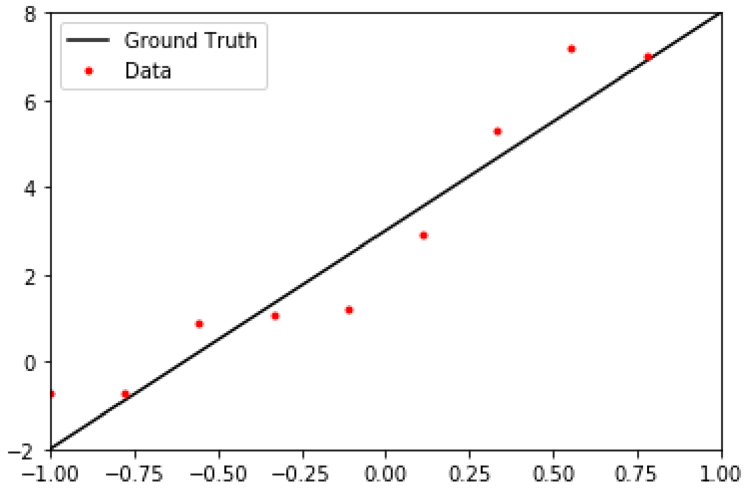


- 1-D linear regression problem
  - Given one dimensional data, find the most fitted line
- Goal
  - Construct linear model in tensorflow graph
  - Define loss function
  - Optimize it using gradient descent method

[Exercise 1] Univariate Linear Regression

1D data generation ¶

```
def f1(x,a,b):
    return a*x+b

a = 5.
b = 3.

x = np.linspace(-1,1,100)
y = f1(x,a,b)

n = 10
x_data = np.linspace(-1,1,n)
y_data = f1(x_data,a,b) + np.random.randn(n)

plt.plot(x,y,"k-",label="Ground Truth")
plt.plot(x_data,y_data,"r.",label="Data")
plt.legend()
plt.xlim([-1,1])
plt.ylim([-2,8])
plt.show()
```



- Step 1
  - Generate 1d noise data from the underlying linear model
  - Plot it

[Exercise 1] Univariate Linear Regression

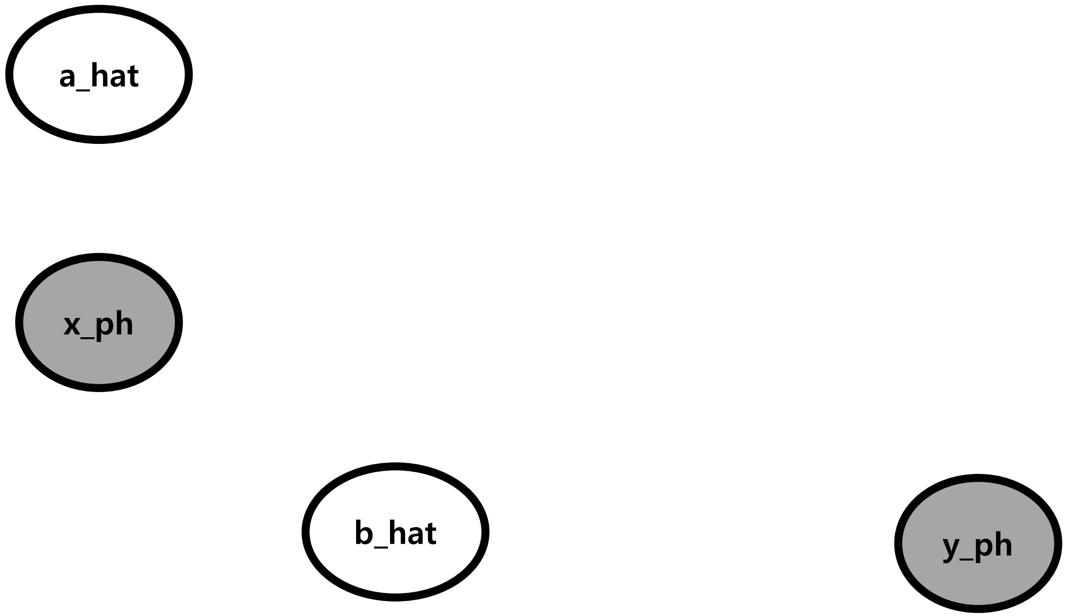
- Step 2
  - Define placeholder : x_ph, y_ph
  - tf.placeholder(dtype, shape=(None,))

Tensorflow Graph Construction

```
tf.reset_default_graph()

x_ph =
y_ph =

a_hat =
b_hat =

y_pred =

loss =

# define optimizer
optimizer =
train =

print("Tensorflow graph is contructed!!!")
print("x_ph is:")
print("type: "+str(type(x_ph)))
print("shape: "+str(x_ph.shape))
print("a_hat is:")
print("type: "+str(type(a_hat)))
print("shape: "+str(a_hat.shape))
print("y_pred is:")
print("type: "+str(type(y_pred)))
print("shape: "+str(y_pred.shape))
```

[Exercise 1] Univariate Linear Regression

- Step 3
  - Define tensorflow variable : a_hat, b_hat
  - tf.get_variable(name, shape=(), initializer=?)
  - tf.random_normal_initializer()

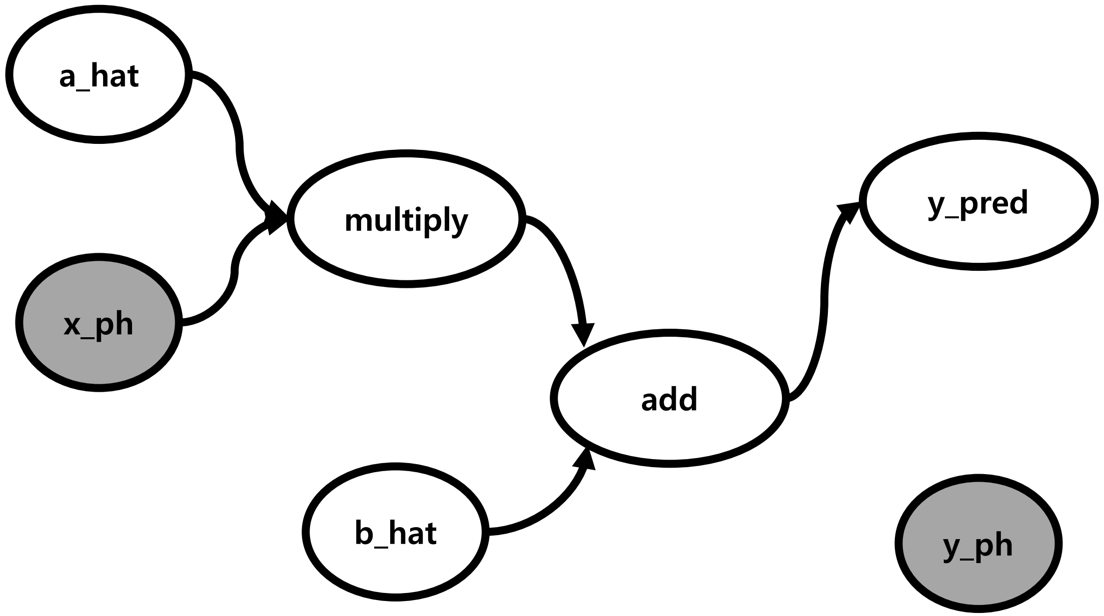


Tensorflow Graph Construction

```
tf.reset_default_graph()

x_ph =
y_ph =

a_hat =
b_hat =

y_pred =

loss =

# define optimizer
optimizer =
train =

print("Tensorflow graph is contructed!!!")
print("x_ph is:")
print("type: "+str(type(x_ph)))
print("shape: "+str(x_ph.shape))
print("a_hat is:")
print("type: "+str(type(a_hat)))
print("shape: "+str(a_hat.shape))
print("y_pred is:")
print("type: "+str(type(y_pred)))
print("shape: "+str(y_pred.shape))
```

[Exercise 1] Univariate Linear Regression

- Step 4
  - Define linear model : y_pred
  - tf.multiply(x_ph,a_hat) + b_hat



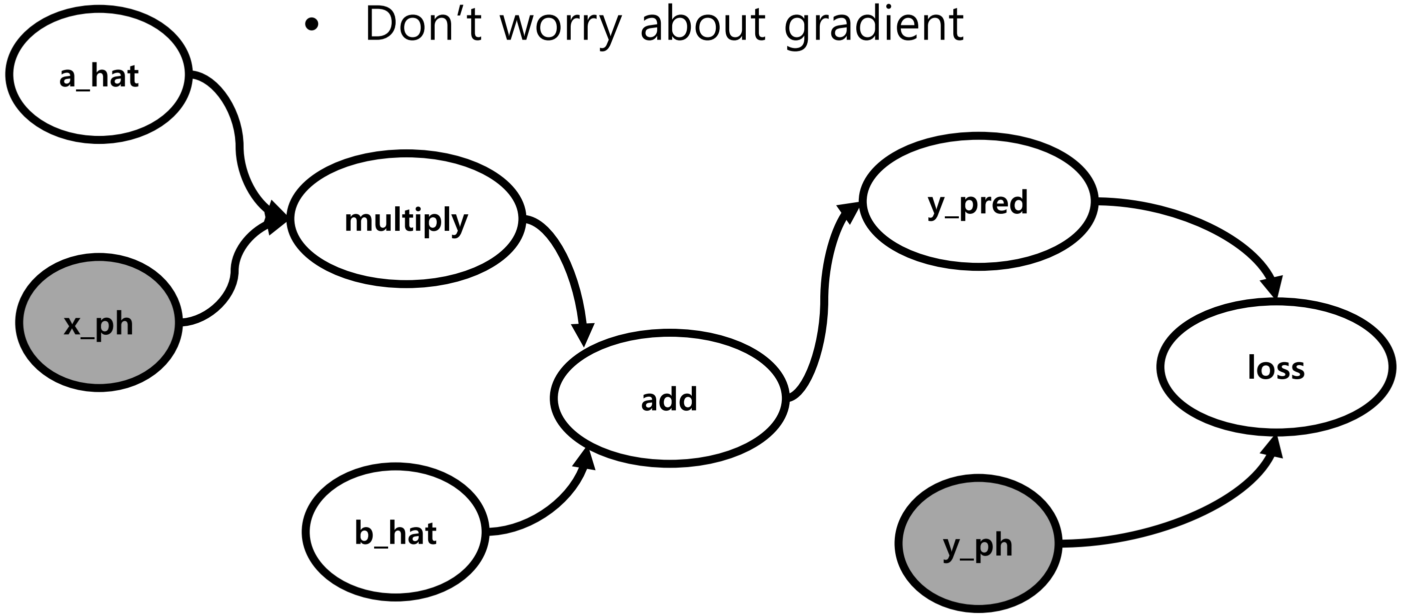
Tensorflow Graph Construction

```
tf.reset_default_graph()

x_ph =
y_ph =

a_hat =
b_hat =

y_pred =

loss =

# define optimizer
optimizer =
train =

print("Tensorflow graph is contructed!!!")
print("x_ph is:")
print("type: "+str(type(x_ph)))
print("shape: "+str(x_ph.shape))
print("a_hat is:")
print("type: "+str(type(a_hat)))
print("shape: "+str(a_hat.shape))
print("y_pred is:")
print("type: "+str(type(y_pred)))
print("shape: "+str(y_pred.shape))
```

[Exercise 1] Univariate Linear Regression

- Step 5
  - Define loss function : loss
  - 0.5 * tf.reduce_mean(tf.square(y_pred - y_ph))

  - Define optimizer : optimizer, train
  - tf.train.GradientDescentOptimizer(learning_rate=0.1)
  - optimizer.minimize(loss)
    - Don't worry about gradient



Tensorflow Graph Construction

```
tf.reset_default_graph()

x_ph =
y_ph =

a_hat =
b_hat =

y_pred =

loss =

# define optimizer
optimizer =
train =

print("Tensorflow graph is contructed!!!")
print("x_ph is:")
print("type: "+str(type(x_ph)))
print("shape: "+str(x_ph.shape))
print("a_hat is:")
print("type: "+str(type(a_hat)))
print("shape: "+str(a_hat.shape))
print("y_pred is:")
print("type: "+str(type(y_pred)))
print("shape: "+str(y_pred.shape))
```

[Exercise 1] Univariate Linear Regression

- Step 6
  - Initialize variable
    - tf.global_variables_initializer()
  - Open tensorflow session
    - tf.Session

Initialize TF variables

```
# initialize variables
init =

sess =
sess.run(init)

a_np, b_np = sess.run([a_hat,b_hat])
print("Randomly initialized parameters")
print("a: %f"%a_np)
print("b: %f"%b_np)
```

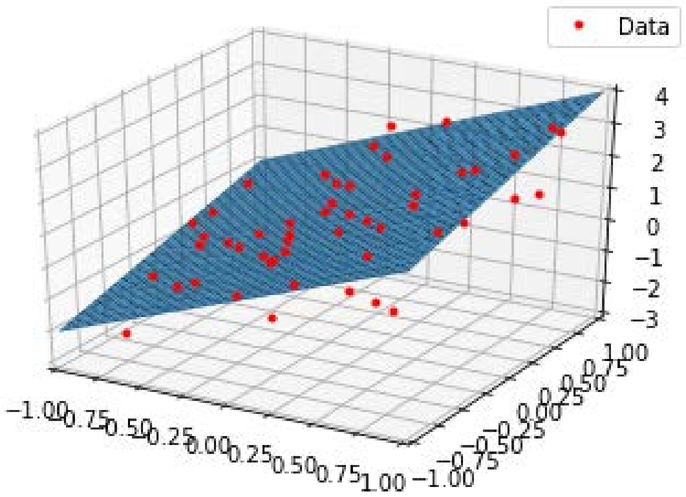
- Step 7
  - Optimize it!
  - Flow the data into placeholder

  - Run optimization
  - sess.run([loss,train],feed_dict=feed_dict)

Optimize linear model (Find the best parameters)

```
nepoch = 3000
feed_dict = {x_ph:,y_ph:}
for epoch in range(nepoch):
    loss_np,_ = |
    if (epoch%300) == 0:
        print("[%d/%d] loss : %f"%(epoch,nepoch,loss_np))
print()

a_np, b_np = sess.run([a_hat,b_hat])
print("Optimized parameters")
print("a: %f"%a_np)
print("b: %f\n"%b_np)
print("Original parameters")
print("a: %f"%a)
print("b: %f"%b)

feed_dict = {x_ph:x}
y_pred_np = sess.run(y_pred,feed_dict=feed_dict)
plt.plot(x,y,"k-",label="Ground Truth")
plt.plot(x_data,y_data,"r.",label="Data")
plt.plot(x,y_pred_np,"b-",label="Linear Regression")
plt.legend()
plt.xlim([-1,1])
plt.ylim([-2,8])
plt.show()
```

[Exercise 2] Multivariate Linear Regression



- Do the same thing for two dimensional data
- Goal
  - Construct linear model in tensorflow graph
  - Define loss function
  - Optimize it using gradient descent method

2D data generation

```
def f2(x,a,b):
    return np.matmul(x,a)+b

a = np.array([[2.],[1.]])
b = 1.

x1 = np.linspace(-1,1,100)
x2 = np.linspace(-1,1,100)
X1,X2 = np.meshgrid(x1,x2)
x = np.concatenate([np.reshape(X1,[-1,1]),np.reshape(X2,[-1,1])],axis=1)
y = f2(x,a,b)
Y = np.reshape(y,[100,100])

n = 50
x_data = np.random.uniform(-1,1,size=(n,2))
y_data = f2(x_data,a,b) + np.random.randn(n,1)

fig = plt.figure()
ax = fig.gca(projection='3d')

ax.plot_surface(X1,X2,Y)
ax.plot3D(x_data[:,0].flatten(),x_data[:,1].flatten(),y_data.flatten(),'r.',label="Data")
plt.legend()
ax.set_xlim3d(-1,1)
ax.set_ylim3d(-1,1)
ax.set_zlim3d(-3,4)
plt.show()
```

[Exercise 2] Multivariate Linear Regression

- Step 1
  - Define placeholder
  - tf.placeholder(dtype, shape=(None,2))
  - tf.placeholder(dtype, shape=(None,1))
- Step 2
  - Define tensorflow variable
  - tf.get_variable(name, shape=(2,1), initializer=?)
  - tf.random_normal_initializer()
- Step 3
  - Define linear model
  - tf.multiply(x_ph,a_hat) + b_hat
- Step 4
  - Define loss function
  - 0.5 * tf.reduce_mean(tf.square(y_pred - y_ph))
  - Define optimizer
- Step 5
  - Run optimization

Tensorflow Graph Construction

Initialize graph

Train it

```
tf.reset_default_graph()

x_ph =
y_ph =

a_hat =
b_hat =

y_pred =

loss =

# define optimizer
optimizer =
train =

# initialize variables
init = tf.global_variables_initializer()
```

```
nepoch = 10000
feed_dict = {x_ph:,y_ph:}
for epoch in range(nepoch):
    loss_np,_ =
    if (epoch%1000) == 0:
        print("[%d/%d] loss : %f"%(epoch,nepoch,loss_np))
```

[Exercise 3] General Linear Regression

- Linear model: $h_{\mathbf{w}}(\mathbf{x}_j) = \sum_{i=0}^{n} w_i x_{j,i} = \mathbf{w}^T \mathbf{x}_j$.
- General linear model:

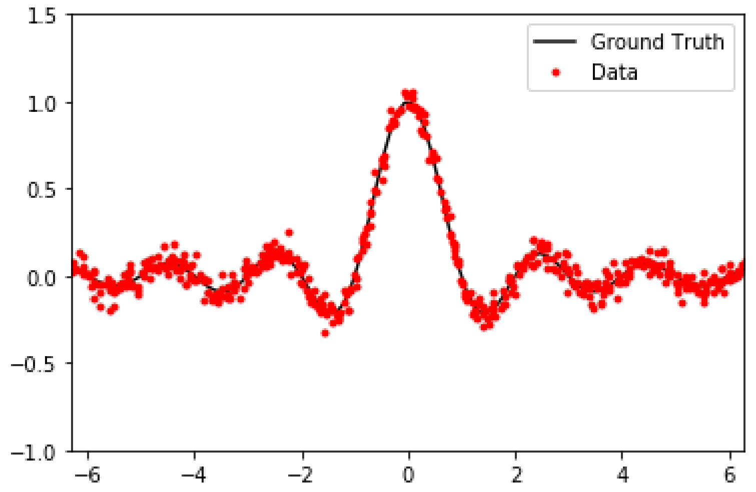
$$h_{\mathbf{w}}(\mathbf{x}_j) = \sum_{i=0}^{n} w_i \phi_i(\mathbf{x}_j) = \mathbf{w}^T \phi(\mathbf{x}_j),$$

where $\phi(\mathbf{x}_j) = [1\ \phi_1(\mathbf{x}_j) \cdots \phi_n(\mathbf{x}_j)]^T$ and $\phi_i : \mathbb{R}^n \to \mathbb{R}$ are nonlinear functions.

- Examples

$$\begin{aligned} \phi_i(x) &= x^i \qquad \text{(polynomial regression)} \\ \phi_i(\mathbf{x}) &= \exp\left(-\frac{1}{2s^2}(\mathbf{x} - \mu_i)^2\right), \end{aligned}$$

where s and μ_i are fixed parameters.



Non-linear 1D data generation

```
def f3(x):
    return np.sinc(x)

x = np.linspace(-2*np.pi,2*np.pi,100)
y = f3(x)

n = 500
x_data = np.linspace(-2*np.pi,2*np.pi,n) + 0.05*np.random.randn(n)
y_data = f3(x_data) + 0.05*np.random.randn(n)

plt.plot(x,y,"k-",label="Ground Truth")
plt.plot(x_data,y_data,"r.",label="Data")
plt.legend()
plt.xlim([-2*np.pi,2*np.pi])
plt.ylim([-1,1.5])
plt.show()
```

[Exercise 3] General Linear Regression

- General linear model: $h_{\mathbf{w}}(\mathbf{x}_j) = \mathbf{w}^T \phi(\mathbf{x}_j)$
- Let

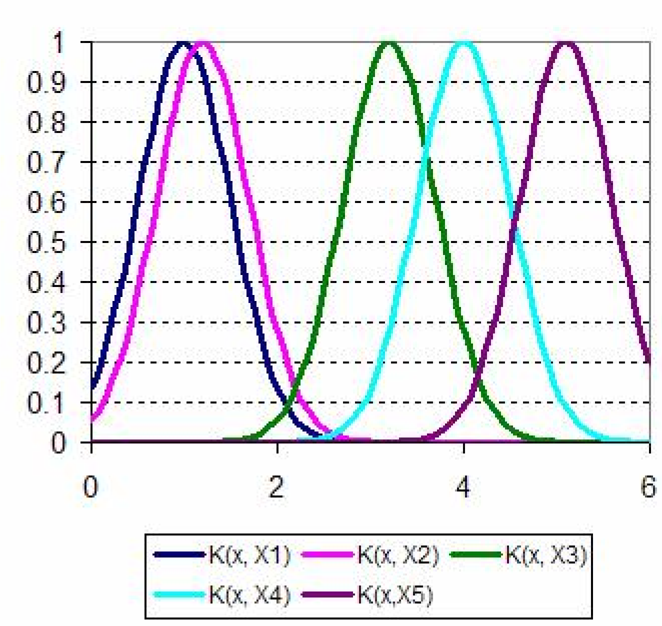
$$\mathbf{\Phi} = \begin{bmatrix} 1 & \phi_1(\mathbf{x}_1) & \cdots & \phi_n(\mathbf{x}_1) \\ 1 & \phi_1(\mathbf{x}_2) & \cdots & \phi_n(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \phi_1(\mathbf{x}_N) & \cdots & \phi_n(\mathbf{x}_N) \end{bmatrix}.$$

- Then

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N} (y_j - \mathbf{w}^T \phi(\mathbf{x}_j))^2 = (\mathbf{y} - \mathbf{\Phi}\mathbf{w})^T (\mathbf{y} - \mathbf{\Phi}\mathbf{w}).$$

- Now, we use matrix form!
- Goal
  - Construct the matrix Φ in tensorflow graph
  - Define predictor and loss function
  - Optimize weight parameters

Example of $\phi_i(x)$



$$\phi_i(\mathbf{x}) = \exp\left(-\frac{1}{2s^2}(\mathbf{x} - \mu_i)^2\right)$$

[Exercise 3] General Linear Regression

- The matrix Φ

Compute this first!

$$\Phi = \begin{bmatrix} 1 & \phi_1(\mathbf{x}_1) & \cdots & \phi_n(\mathbf{x}_1) \\ 1 & \phi_1(\mathbf{x}_2) & \cdots & \phi_n(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \phi_1(\mathbf{x}_N) & \cdots & \phi_n(\mathbf{x}_N) \end{bmatrix}$$

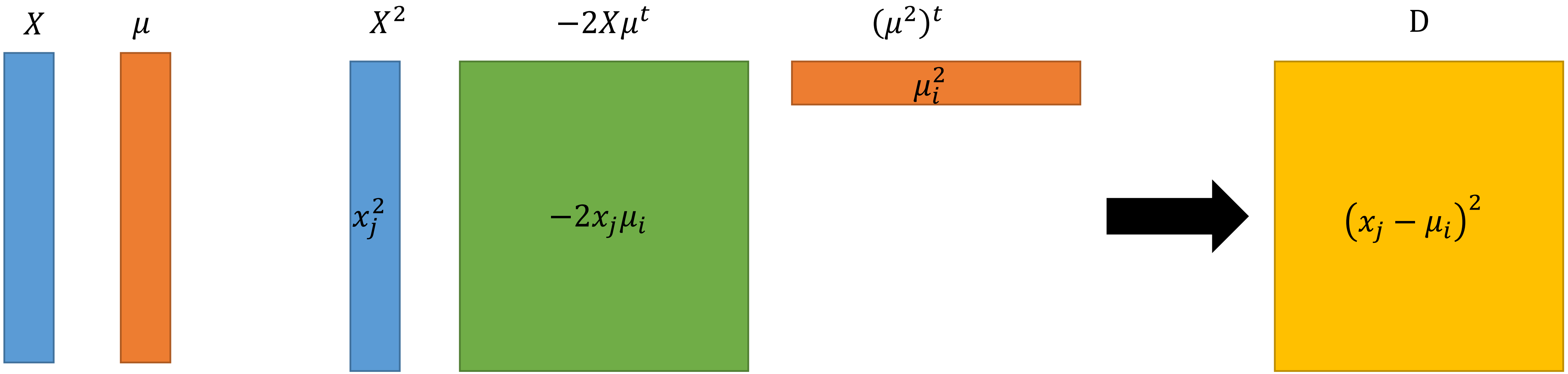
The number of data : N

The number of kernel function ϕ_i : n

- All elements look like $\phi_i(x_j) = \exp\left(-\frac{(x_j-\mu_i)^2}{2s^2}\right)$
  - Compute distance matrix
    - $D_{ij} = \left[(x_j - \mu_i)^2\right]$
  - Compute kernel matrix using element-wise operation
    - $\Phi = \left[1, \quad \exp\left(-\frac{D}{2s^2}\right)\right]$

[Exercise 3] General Linear Regression

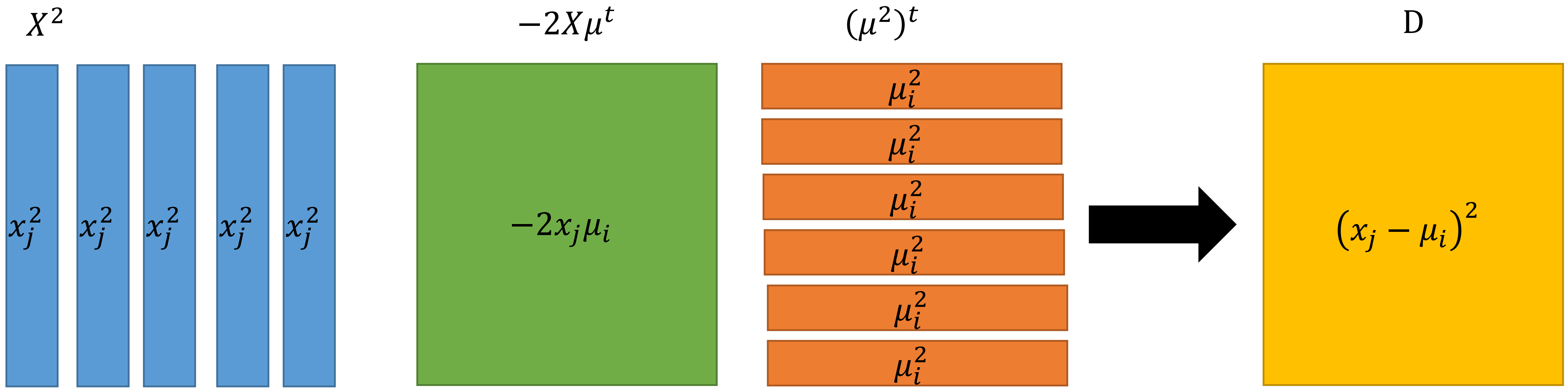
- The matrix D



  - Python broadcasting!
  - TF broadcasting!

[Exercise 3] General Linear Regression

- The matrix D



  - Python broadcasting!
  - TF broadcasting!

[Exercise 3] General Linear Regression

- Step 1
  - Define placeholder
  - x_ph : shape (None,1)
  - y_ph : shape (None,1)
  - mu_ph : μ_i
    - shape (n_kernel+1, 1)
  - inv_squared_s_ph : $h = s^{-2}$
    - shape ()
  - w_hat : w_i
    - shape (n_kernel+1, 1)

- Step 2
  - Compute X^2 : tf.reduce_sum(tf.square(x_ph), axis=1)
  - Compute μ^2
  - Reshape X^2 : tf.reshape(x_norm, [-1, 1])
  - Reshape μ^2
  - $D = X^2 - 2X\mu^t + \mu^2$: tf.matmul(x_ph, mu_ph, False, True)

Tensorflow Graph Construction

```
tf.reset_default_graph()

n_kernel = 20
mu = np.linspace(-2*np.pi,2*np.pi,n_kernel)
inv_squared_s = 1e1

x_ph =
y_ph =

mu_ph =
inv_squared_s_ph =

w_hat =
print("Define place holders")
```

Distance Matrix D

$X^2 - 2X\mu' + \mu^2$

```
x_norm =
mu_norm =

x_norm =
mu_norm =

squared_dist =
```

[Exercise 3] General Linear Regression

- Step 3
  - Compute kernel matrix
    - $\exp\left(-\frac{D}{2s^2}\right)$
  - Concatenate one vector
    - $\Phi = \left[1, \quad \exp\left(-\frac{D}{2s^2}\right)\right]$
- Step 4

```
kernel =
kernel =

y_pred = tf.matmul(kernel,w_hat)

loss = 0.5 * tf.reduce_mean( tf.square(y_pred - y_ph) )

optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
train = optimizer.minimize(loss)
```

Train

```
init = tf.global_variables_initializer()

sess = tf.Session()
sess.run(init)

w_np = sess.run(w_hat)
print("Randomly initialized parameters")
print("w: {:}₩n".format(w_np.flatten()))

nepoch = 10000
feed_dict = {x_ph:x_data[:,np.newaxis],y_ph:y_data[:,np.newaxis],mu_ph:mu[:,np.newaxis],inv_squared_s_ph:inv_squared_s}

for epoch in range(nepoch):
    loss_np,_ = sess.run([loss,train],feed_dict=feed_dict)
    if (epoch%1000) == 0:
        print("[%d/%d] loss : %f"%(epoch,nepoch,loss_np))
```