In [1]:
```
n_repeats = 5
n_splits = 10
n_epochs = 1000
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# 回歸問題不能用stratifiedKFold
from sklearn.model_selection import RepeatedKFold
import sklearn.metrics
import numpy as np
```

In [4]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import regularizers
import matplotlib.pyplot as plt
```

In [5]:
```
from sklearn.datasets import load_diabetes
(X, y) = load_diabetes(return_X_y=True, as_frame=True)
X.head()
# data是被normalize過的
```

Out[5]:

	age	sex	bmi	bp	s1	s2	s3	s4	s5
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019908
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068330
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002864
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022692
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031991

In [6]:
```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(handle_unknown='ignore')
def one_hot(x):
    return enc.fit_transform(x.reshape(-1, 1)).toarray()
```

In [7]:
```
X_nbp = np.array(X.drop(['bp'], axis=1))
sex = np.array(one_hot(np.array(X['sex'])))
s = np.array(X.iloc[:, 4:])
ab = np.array(X[['age', 'bmi']])
abb = np.array(X[['age', 'bmi', 'bp']])
y = np.array(y)
X = np.array(X)

# 不考慮血壓，找一個最好的model
# sequential, autokeras: X_nbp
# funtional: ("ab" + s + sex)

# 考慮血壓，找一個最好的model
# sequential, autokeras: X
# funtional: ("abb" + s + sex)
```

In [8]:
```
sex[0]
```

Out[8]:
```
array([0., 1.])
```

model_0(sequential)跟model_1(functional)吃沒有血壓的feature，挑一個比較好的model

In [9]:
```
model_0 = tf.keras.Sequential(name="sequentialNbp")
model_0.add(layers.Dense(1,
                         input_shape=(X_nbp.shape[1], )))
```

In [10]:
```
ab_in_m1 = keras.Input(shape=(ab.shape[1],), name='ab_in_m1')
ab_h1_m1 = layers.Dense(3, activation='relu',\
                        kernel_regularizer=\
                       keras.regularizers.l1_l2(l1=0.1, l2=0.01))(ab_in_m1)
o_1_m1 = layers.Dense(2)(ab_h1_m1)
s_in_m1 = keras.Input(shape=(s.shape[1],), name='s_in_m1')
s_h1_m1 = layers.Dense(4, activation='relu',\
                        kernel_regularizer=\
                      keras.regularizers.l1_l2(l1=0.1, l2=0.01))(s_in_m1)
o_2_m1 = layers.Dense(4)(s_h1_m1)
sex_m1 = keras.Input(shape=(sex.shape[1],), name='sex_m1')
o_3_m1 = layers.Dense(1)(sex_m1)
concat_m1 = layers.Concatenate()([o_1_m1, o_2_m1, o_3_m1])
h1_m1 = layers.Dense(4, name='h1_m1')(concat_m1)
outputs_m1 = layers.Dense(1, activation='linear', name='y')(h1_m1)
model_1 = keras.Model(inputs=[ab_in_m1, s_in_m1, sex_m1], \
                      outputs=outputs_m1, name='functionalNbp')
print("model_1(tf functional API)", model_1.summary())
```

```
Model: "functionalNbp"
__________________________________________________________________________________
_________________
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================
==================
ab_in_m1 (InputLayer)        [(None, 2)]               0
__________________________________________________________________________________
_________________
s_in_m1 (InputLayer)         [(None, 6)]               0
__________________________________________________________________________________
_________________
dense_1 (Dense)              (None, 3)                 9           ab_in_m1[0][0]
__________________________________________________________________________________
_________________
dense_3 (Dense)              (None, 4)                 28          s_in_m1[0][0]
__________________________________________________________________________________
_________________
sex_m1 (InputLayer)          [(None, 2)]               0
__________________________________________________________________________________
_________________
dense_2 (Dense)              (None, 2)                 8           dense_1[0][0]
__________________________________________________________________________________
_________________
dense_4 (Dense)              (None, 4)                 20          dense_3[0][0]
__________________________________________________________________________________
_________________
dense_5 (Dense)              (None, 1)                 3           sex_m1[0][0]
__________________________________________________________________________________
_________________
concatenate (Concatenate)    (None, 7)                 0           dense_2[0][0]
                                                                   dense_4[0][0]
                                                                   dense_5[0][0]
__________________________________________________________________________________
_________________
h1_m1 (Dense)                (None, 4)                 32          concatenate[0]
[0]
```

```
_________________________________________________________________
y (Dense)                    (None, 1)                 5         h1_m1[0][0]
=================================================================
Total params: 105
Trainable params: 105
Non-trainable params: 0
_________________________________________________________________
model_1(tf functional API) None
```

In [11]:
```
model_2 = tf.keras.Sequential(name="sequentialWbp")
model_2.add(layers.Dense(1,
                         input_shape=(X.shape[1], )))
```

model_2 3 是仿照0 1 的架構 但是feature有含血壓的 也挑一個比較好的model

In [12]:
```
abb_in_m3 = keras.Input(shape=(abb.shape[1],), name='abb_in_m3')
abb_h1_m3 = layers.Dense(3, activation='relu',\
                         kernel_regularizer=\
                         keras.regularizers.l1_l2(l1=0.1, l2=0.01))(abb_in_m3)
o_1_m3 = layers.Dense(3)(abb_h1_m3)
s_in_m3 = keras.Input(shape=(s.shape[1],), name='s_in_m3')
s_h1_m3 = layers.Dense(4, activation='relu',\
                         kernel_regularizer=\
                        keras.regularizers.l1_l2(l1=0.1, l2=0.01))(s_in_m3)
o_2_m3 = layers.Dense(4)(s_h1_m3)
sex_m3 = keras.Input(shape=(sex.shape[1],), name='sex_m3')
o_3_m3 = layers.Dense(1)(sex_m3)
concat_m3 = layers.Concatenate()([o_1_m3, o_2_m3, o_3_m3])
h1_m3 = layers.Dense(3, name='h1_m3')(concat_m3)
outputs_m3 = layers.Dense(1, activation='linear', name='y')(h1_m3)
model_3 = keras.Model(inputs=[abb_in_m3, s_in_m3, sex_m3], \
                      outputs=outputs_m3, name='functionalWbp')
print("model_3(tf functional API)", model_3.summary())
```

```
Model: "functionalWbp"
__________________________________________________________________________________
_________________
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================
==================
abb_in_m3 (InputLayer)       [(None, 3)]               0
__________________________________________________________________________________
_________________
s_in_m3 (InputLayer)         [(None, 6)]               0
__________________________________________________________________________________
_________________
dense_7 (Dense)              (None, 3)                 12          abb_in_m3[0]
[0]
__________________________________________________________________________________
_________________
dense_9 (Dense)              (None, 4)                 28          s_in_m3[0][0]
__________________________________________________________________________________
_________________
sex_m3 (InputLayer)          [(None, 2)]               0
__________________________________________________________________________________
_________________
dense_8 (Dense)              (None, 3)                 12          dense_7[0][0]
__________________________________________________________________________________
_________________
dense_10 (Dense)             (None, 4)                 20          dense_9[0][0]
__________________________________________________________________________________
_________________
dense_11 (Dense)             (None, 1)                 3           sex_m3[0][0]
__________________________________________________________________________________
_________________
concatenate_1 (Concatenate)  (None, 8)                 0           dense_8[0][0]
                                                                   dense_10[0][0]
                                                                   dense_11[0][0]
__________________________________________________________________________________
_________________
```

```
h1_m3 (Dense)                (None, 3)             27          concatenate_1
[0][0]
__________________________________________________________________________________
________________
y (Dense)                    (None, 1)             4           h1_m3[0][0]
==================================================================================
==================
Total params: 106
Trainable params: 106
Non-trainable params: 0
__________________________________________________________________________________
________________
model_3(tf functional API) None
```

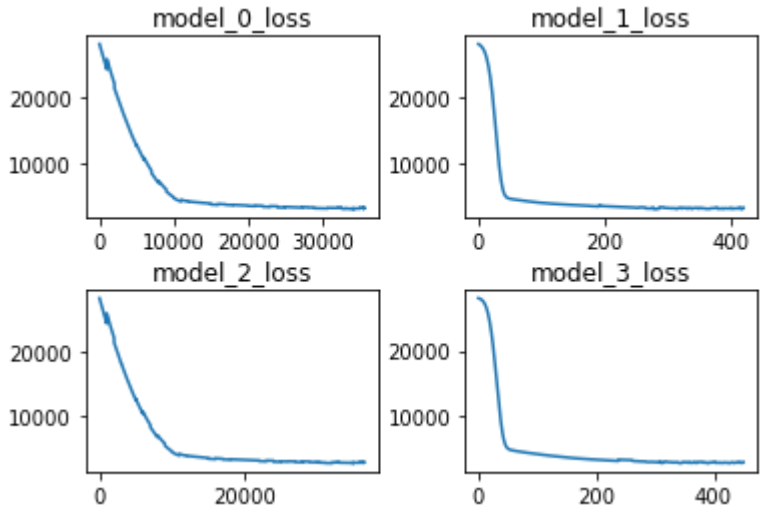
In [13]:
```
# 回歸問題用MSE
n_models = 4
for i in range(n_models):
    locals()['model_{}'.format(i)].compile(
            optimizer='adam',
            loss=tf.keras.losses.MeanSquaredError())
    locals()['loss_model_{}'.format(i)] = []
    locals()['val_MSE_{}'.format(i)] = []
```

In [14]:
```
%%time
early_stop = tf.keras.callbacks.EarlyStopping\
(monitor='loss', patience=1, min_delta=1e-4)
rskf = RepeatedKFold(n_repeats=n_repeats, n_splits=n_splits)
for train_index, test_index in rskf.split(X, y):

    # 切training testing data

    # 考慮血壓，找一個最好的model
    # sequential(model_0), autokeras(model_4): X_nbp
    # funtional(model_1): ("ab" + s + sex)

    # 不考慮血壓，找一個最好的model
    # sequential(model_2), autokeras(model_5): X
    # funtional(model_3): ("abb" + s + sex)
    X_tr, X_te = X[train_index], X[test_index]
    y_tr, y_te = y[train_index], y[test_index]
    ab_tr, s_tr, sex_tr = ab[train_index], s[train_index], sex[train_index]
    ab_te, s_te, sex_te = ab[test_index], s[test_index], sex[test_index]
    abb_tr, abb_te = abb[train_index], abb[test_index]
    X_nbp_tr, X_nbp_te = X_nbp[train_index], X_nbp[test_index]


    ###fit models###
    # 0 1  是不吃血壓的model   2 3是吃血壓的model
    history_0 = model_0.fit(X_nbp_tr, y_tr, epochs=n_epochs, \
                            verbose=0, callbacks=[early_stop])
    history_1 = model_1.fit(\
        {'ab_in_m1': ab_tr, 's_in_m1': s_tr, 'sex_in_m1': sex_tr},
        {'y': y_tr}, epochs=n_epochs, \
                            verbose=0, callbacks=[early_stop])

    history_2 = model_2.fit(X_tr, y_tr, epochs=n_epochs, \
                            verbose=0, callbacks=[early_stop])
    history_3 = model_3.fit(\
        {'abb_in_m3': abb_tr, 's_in_m3': s_tr, 'sex_in_m3': sex_tr},
        {'y': y_tr}, epochs=n_epochs, \
                            verbose=0, callbacks=[early_stop])

    ###拿到各model的prediction來算MSE
    pred_0 = model_0.predict(X_nbp_te)
    pred_1 = model_1.predict(
        {'ab_in_m1': ab_te, 's_in_m1': s_te, 'sex_in_m1': sex_te})
    pred_2 = model_2.predict(X_te)
    pred_3 = model_3.predict(
        {'abb_in_m3': abb_te, 's_in_m3': s_te, 'sex_in_m3': sex_te})

    for i in range(n_models):

        ###把loss存起來###
        locals()['loss_model_{}'.format(i)].extend\
        (locals()['history_{}'.format(i)].history['loss'])

        ###validation data用來evaluate model好壞by MSE###
        locals()['val_MSE_{}'.format(i)].append\
        (sklearn.metrics.mean_squared_error\
```

```
        (y_te, locals()['pred_{}'.format(i)]))
```

```
Wall time: 8min 17s
```

In [15]:
```
plt.subplots_adjust(hspace=0.4, wspace=0.3)
for i in range(n_models):
    plt.subplot(2, 2, i+1)
    plt.title('model_{}_loss'.format(i))
    plt.plot(locals()['loss_model_{}'.format(i)])
```



In [16]:
```
# 把每次cv完的結果取平均
for i in range(n_models):
     locals()['mean_val_MSE_{}'.format(i)] = \
     np.mean(locals()['val_MSE_{}'.format(i)])
     # mean of (n_splits * n_repeats) values
```

In [18]:
```
k = locals()
no_bp_model_MSE = [(k['mean_val_MSE_{}'.format(i)]) for i in range(2)]
no_bp_model_best_idx = np.argmin(no_bp_model_MSE)
no_bp_model_best_MSE = np.min(no_bp_model_MSE)
with_bp_model_MSE = [(k['mean_val_MSE_{}'.format(i)]) for i in range(2, 4)]
with_bp_model_best_idx = np.argmin(with_bp_model_MSE) + 2
with_bp_model_best_MSE = np.min(with_bp_model_MSE)

print('不考慮血壓model裡面，預測能力最好的是model_{}，平均MSE={}。'\
      .format(no_bp_model_best_idx, no_bp_model_best_MSE))
print('考慮血壓model裡面，預測能力最好的是model_{}，平均MSE={}。'\
      .format(with_bp_model_best_idx, with_bp_model_best_MSE))

if no_bp_model_best_MSE < with_bp_model_best_MSE:
      print('考慮血壓的模型有較高的MSE，可知血壓對於糖尿病病程的影響不大，若考慮血壓會降
else:
      print('考慮血壓的模型有較低的MSE，可知血壓這個feature對於預測糖尿病病程有一定的貢
```

```
不考慮血壓model裡面，預測能力最好的是model_1，平均MSE=3133.710642613691。
考慮血壓model裡面，預測能力最好的是model_3，平均MSE=2971.6992045433326。
考慮血壓的模型有較低的MSE，可知血壓這個feature對於預測糖尿病病程有一定的貢獻
```