1. A model having a high bias with very few parameters will

   A. underfit the training data

   B. overfit the training data

**Solution: Option A** is the correct answer

2. A model having a high variance with a large number of parameters will

   A. have a low expected mean square error

   B. have a high expected mean square error

**Solution: Option B** is the correct answer. We saw that the expected mean square error is given by:

$$E[(y - \hat{f}(x))^2] = Bias^2 + Variance + \sigma^2 \textit{(irreducible error)}$$

Hence if the variance is high, the expected mean square error will be high.

3. Is the following statements true: A complex model will be more sensitive to changes in training data than a simple model.

   A. True

   B. False

**Solution: Option A** is the correct answer.

4. Consider a model which has only 3 parameters, $w_1$, $w_2$, $w_3$ or more compactly $\theta = [w_1, w_2, w_3]$. Let $\mathscr{L}(\theta)$ be the loss function and $\Omega(\theta)$ be the regularizer. For example, if we use L2 regularization than $\Omega(\theta) = ||\theta||^2$. The corresponding gradient descent update rule for $w_1$ is given by

$$w_1 = w_1 - \eta \frac{\partial}{\partial w_1} \mathscr{L}(w_1) - 2\eta\alpha w_1$$

and we will have a similar equation for $w_2$ and $w_3$ also. Suppose, instead of L2 regularization we use L4 regularization then what would the update rule for $w_1$ be

A. $w_1 = w_1 - \eta \frac{\partial}{\partial w_1} \mathscr{L}(w_1) - \eta\alpha w_1$

B. $w_1 = w_1 - \eta \frac{\partial}{\partial w_1} \mathscr{L}(w_1) - \eta\alpha$

C. $w_1 = w_1 - \eta \frac{\partial}{\partial w_1} \mathscr{L}(w_1) - 3\eta\alpha w_1^2$

D. $w_1 = w_1 - \eta \frac{\partial}{\partial w_1} \mathscr{L}(w_1) - 4\eta\alpha w_1^3$

**Solution: Option D** is the correct answer. The total loss function is given by:

$$\begin{aligned} \mathscr{L}_{total}(\theta) &= \mathscr{L}(\theta) + \Omega(\theta) \\ &= \mathscr{L}(\theta) + ||\theta||^4 \\ &= \mathscr{L}(\theta) + w_1^4 + w_2^4 + w_3^4 \end{aligned}$$

Taking a partial derivative w.r.t $w_1$ we get,

$$\begin{aligned} \frac{\partial}{\partial w_1} \mathscr{L}_{total}(\theta) &= \frac{\partial}{\partial w_1} \mathscr{L}(\theta) + \frac{\partial}{\partial w_1} w_1^4 + \frac{\partial}{\partial w_1} w_2^4 + \frac{\partial}{\partial w_1} w_3^4 \\ &= \frac{\partial}{\partial w_1} \mathscr{L}(\theta) + 4w_1^3 \end{aligned}$$

Now the gradient descent update rule is,

$$\begin{aligned} w_1 &= w_1 - \eta \frac{\partial}{\partial w_1} \mathscr{L}_{total}(w_1) \\ &= w_1 - \eta \frac{\partial}{\partial w_1} \mathscr{L}(w_1) - 4\eta\alpha w_1^3 \end{aligned}$$

5. Suppose you are training a simple neural network as defined by the equations below:

$$y = f(x) \\ = \frac{1}{1 + e^{-}(w^T x)}$$

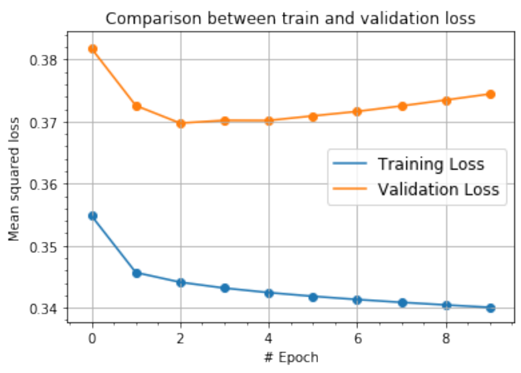
where $x \in \mathbb{R}^{40}, w \in \mathbb{R}^{40}$. The dataset is divided into a training set and a validation set. We have trained the network for 10 epochs and the weights after each epoch are downloadable. You can load these weights using the following code:

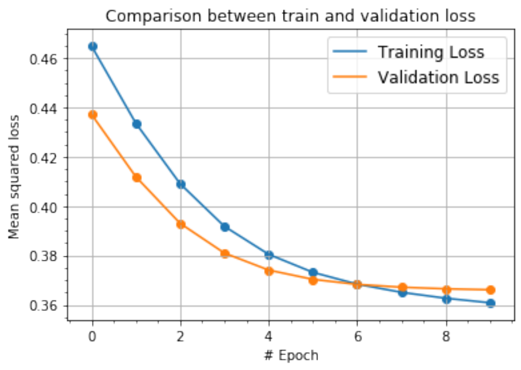
```python
import numpy as np
# Before epoch 0, load this
w = np.load('weights_after_epoch_0.npy')
# Before epoch 1, load this
w = np.load('weights_after_epoch_1.npy')
# .. and so on till
# Before epoch 9, load this
w = np.load('weights_after_epoch_9.npy')
```

You need to compute the training and validation error after each epoch and say which of the following plots correctly shows the training and validation error. We have used mean squared loss function while training the network. You can calculate the loss for each individual data-point using the definition of "error" as defined in the code snippets (Slide 36/62 in Week 2 study material) and the total loss after every epoch is calculated by the taking the mean of these individual losses.

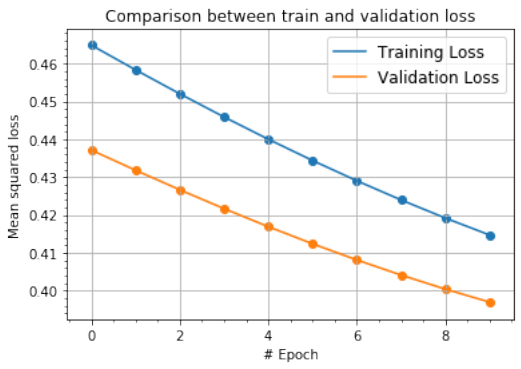
You can download the data using the url:
https://drive.google.com/drive/folders/1gw26BEHSuwx3Enp300Rr1u0rshrhHRA8?usp=sharing

You can download the weights after every epoch using the url:
https://drive.google.com/drive/folders/1-yXdSiYviVtpPnEp0V61R_ZNmIPpKkGA?usp=sharing

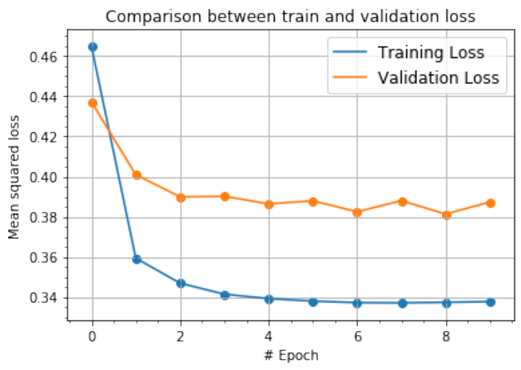
A. 



B.



C.



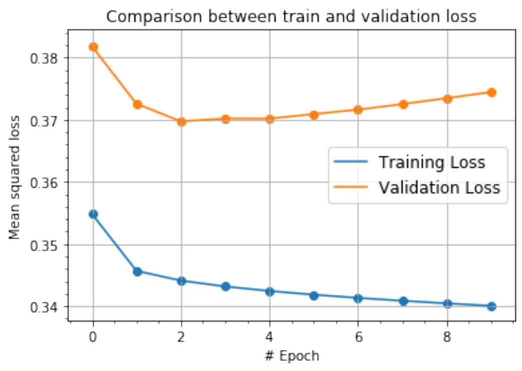
D.

**Solution:** You can use the following code to generate the above plots :

```
# IMPORT PACKAGES
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# DEFINE SIGMOID FUNCTION
def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))

```

```
# DEFINE PLOT FUNCTION
def plot_loss(training_loss,validation_loss,max_epochs):
    # PLOT THE TRAINING LOSS
    x_axis = list(range(0, max_epochs))
    y_axis = training_loss

    plt.plot(x_axis, y_axis , label='Training Loss')
    plt.scatter(x_axis, y_axis, label=None)

    # PLOT THE VALIDATION LOSS
    x_axis = list(range(0, max_epochs))
    y_axis = validation_loss

    plt.plot(x_axis, y_axis , label='Validation Loss')
    plt.scatter(x_axis, y_axis, label=None)

    plt.xlabel("# Epoch")
    plt.ylabel("Mean squared loss")
    plt.legend(prop={'size' : 12}) # prop is to set the font properties
     of the legend
    plt.grid()
    plt.minorticks_on()
    plt.title('Comparison between train and validation loss')
    plt.show()

# LOAD TRAINING DATA
def load_compute_error(training_csv,validation_csv):

    df = pd.read_csv(training_csv)

    # SPLIT TRAINING DATA INTO X AND Y
    train_X = df.iloc[:,0:40].values
    train_Y = df.iloc[:,-1].values

    # LOAD VALIDATION DATA
    df = pd.read_csv(validation_csv)

    # SPLIT VALIDATION DATA INTO X AND Y
    val_X = df.iloc[:,0:40].values
    val_Y = df.iloc[:,-1].values

    max_epochs = 10
    current_epoch = 0
    training_loss = []
    validation_loss = []
    while current_epoch < max_epochs:
        # LOADING WEIGHTS BEFORE EVERY EPOCH
        w = np.load('weights_after_epoch_' + str(current_epoch) + '.npy
    ')
        #TRAIN_LOSS
        train_loss = []
        for i, x_i in enumerate(train_X):
            loss = 0
            # FORWARD-PASS
            # GET PREDICTED VALUE
            y_hat_i = sigmoid(np.dot(w, x_i)[0])

            # GET TRUE VALUE
            y_i = train_Y[i]

            # CALCULATE THE MSE LOSS
            loss = 0.5 * np.square(y_hat_i - y_i)
            train_loss.append(loss)

        training_loss.append(np.mean(train_loss))

        # CALCULATE VALIDATION LOSS
        val_loss = []
        for i, x_i in enumerate(val_X):
            loss = 0
            # FORWARD-PASS
            # GET PREDICTED VALUE
            y_hat_i = sigmoid(np.dot(w, x_i)[0])

            # GET TRUE VALUE
            y_i = val_Y[i]

            # CALCULATE THE LOSS
            loss = 0.5 * np.square(y_hat_i - y_i)
            val_loss.append(loss)

        validation_loss.append(np.mean(val_loss))
        print('epoch : {} , Training Loss : {}, Validation Loss : {}'.format(current_epoch, training_loss[current_epoch], validation_loss[current_epoch]))
        current_epoch+=1

    plot_loss(training_loss,validation_loss,max_epochs)

if __name__ == "__main__":

    training_csv = 'training_data.csv'
    validation_csv = 'validation_data.csv'

    load_compute_error(training_csv,validation_csv)
```

We get the plot as follows:



Hence, **Option A** is the correct option.

6. Continuing the previous question, if you decide to use early stopping with a patience of 3 epochs then which model will you pick ?(Note that we number the epochs from epoch 0 to epoch 9)

   A. The model saved after epoch 0

   B. The model saved after epoch 2

   C. The model saved after epoch 4

   D. The model saved after epoch 6

**Solution: Option C** is the correct answer The validation loss is decreasing upto the 4-th epoch but after the 4-th epoch the loss does not decrease for the next 3 epochs. Hence as per the early stopping criteria, you will pick the model saved at the end of 4 epochs.