L5 - Generalization Error

In this exercise wee need to explain all important overall concepts in training. Let's begin with Figure 5.3 from Deep Learning, Ian Goodfellow, et. al. [DL], that pretty much sums it all up

CHAPTER 5. MACHINE LEARNING BASICS

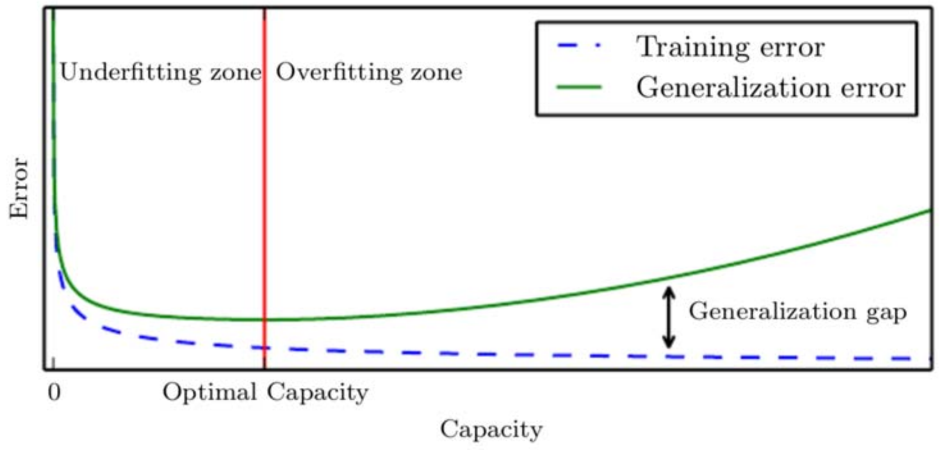


Figure 5.3: Typical relationship between capacity and error. Training and test error behave differently. At the left end of the graph, training error and generalization error are both high. This is the *underfitting regime*. As we increase capacity, training error decreases, but the gap between training and generalization error increases. Eventually, the size of this gap outweighs the decrease in training error, and we enter the *overfitting regime*, where capacity is too large, above the *optimal capacity*.

Qa On Generalization Error

Write a detailed description of figure 5.3 (above) for your journal.

The first part of the graph up until the Optimal Capacity (red line) is what you would call the underfitting zone. The model is simply not getting enough training data. And after the red line is the overfitting zone, where the model simply gets too much training data.

The optimal capacity is the optimal ammount of data the model needs to be trained best. Where fx. with the early stopping you stop when the generalization error reaches its minimum, at the Optimal capacity.

Generalization error describes the models ability to predict unseen instances, but can only be estimated, not computed, as we do not know what data we will recieve in the future. Training error is the sum up of training examples with the predicted values vs the actual values.

The generalization gap occurs when the generalization error starts to climb and eventualy generates a gap between it and the training error. This happens as the model at some point with very large amounts of data starts to degrade its ability to generalize.

Qb A MSE-Epoch/Error Plot

Review the code for plotting the RMSE vs. the iteration number or epoch below (two cells, part I/II).

Write a short description of the code, and comment on the important points in the generation of the (R)MSE array.

The training phase output lots of lines like

```
epoch= 104, mse_train=1.50, mse_val=2.37
epoch= 105, mse_train=1.49, mse_val=2.35
```

What is an *epoch* and what is `mse_train` and `mse_val` ?

epoch is a complete dataset that the model is to be trained by.

`mse_train` is the mean squared error of the training set.

`mse_val` is is the mean squared error of the validation set.

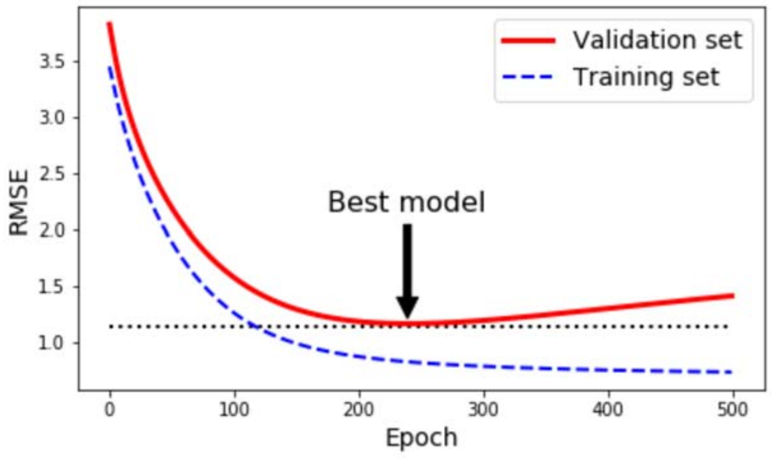
NOTE: the generalization plot in figure 5.3 and the plots below have different x-axis, and are not to be compared directly!

In [17]:
```python
# TODO: Qb(part I)...code review and text

# NOTE: modified code from [GITHOML], 04_training_linear_models.ipynb

%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import SGDRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

np.random.seed(42)

def GenerateData():
    m = 100
    X = 6 * np.random.rand(m, 1) - 3
    y = 2 + X + 0.5 * X**2 + np.random.randn(m, 1)
    return X, y

#Generate some random data to use
X, y = GenerateData()
X_train, X_val, y_train, y_val = train_test_split(X[:50], y[:50].ravel(), test_size
=0.5, random_state=10)

print("X_train.shape=",X_train.shape)
print("X_val  .shape=",X_val.shape)
print("y_train.shape=",y_train.shape)
print("y_val  .shape=",y_val.shape)

#Create pipeline using polynominal features and standard scalar
poly_scaler = Pipeline([
        ("poly_features", PolynomialFeatures(degree=90, include_bias=False)),
        ("std_scaler", StandardScaler()),
    ])

#scale the training data
X_train_poly_scaled = poly_scaler.fit_transform(X_train)
X_val_poly_scaled   = poly_scaler.transform(X_val)


print('OK')
```

```
X_train.shape= (25, 1)
X_val  .shape= (25, 1)
y_train.shape= (25,)
y_val  .shape= (25,)
OK
```

In [18]:
```
# TODO: Qb(part II)...code review and text.
#       NOTE: scroll down in the output to see the best-model figure!

def Train(X_train, y_train, X_val, y_val, n_epochs, verbose=False):
    print("Training...n_epochs=",n_epochs)
    train_errors, val_errors = [], []

    #Use the Stochastic Gradient Decent regressor model
    sgd_reg = SGDRegressor(max_iter=1,
                           penalty=None,
                           eta0=0.0005,
                           warm_start=True,
                           learning_rate="constant",
                           tol=-float("inf"),
                           random_state=42)

    #Train the model on each epoch and get the training, and validation errors.
    for epoch in range(n_epochs):
        sgd_reg.fit(X_train, y_train)
        y_train_predict = sgd_reg.predict(X_train)
        y_val_predict   = sgd_reg.predict(X_val)

        mse_train=mean_squared_error(y_train, y_train_predict)
        mse_val  =mean_squared_error(y_val  , y_val_predict)

        train_errors.append(mse_train)
        val_errors  .append(mse_val)
        #printing the mse for training and validation set at each epoch
        #For saving a bit of space in the journal we'll disable this feature.
        #if verbose:
            #print(f"  epoch={epoch:4d}, mse_train={mse_train:4.2f}, mse_val={mse_v
al:4.2f}")

    return train_errors, val_errors

n_epochs = 500
train_errors, val_errors = Train(X_train_poly_scaled, y_train, X_val_poly_scaled, y
_val, n_epochs, True)

# Calculate where the optimal capacity is, for the getting the best model.
best_epoch = np.argmin(val_errors)
best_val_rmse = np.sqrt(val_errors[best_epoch])

# Now plot where the best model is
plt.figure(figsize=(7,4))
plt.annotate('Best model',
             xy=(best_epoch, best_val_rmse),
             xytext=(best_epoch, best_val_rmse + 1),
             ha="center",
             arrowprops=dict(facecolor='black', shrink=0.05),
             fontsize=16,
            )
#Plot the rest
best_val_rmse -= 0.03  # just to make the graph look better
plt.plot([0, n_epochs], [best_val_rmse, best_val_rmse], "k:", linewidth=2)
plt.plot(np.sqrt(val_errors), "r-", linewidth=3, label="Validation set")
plt.plot(np.sqrt(train_errors), "b--", linewidth=2, label="Training set")
plt.legend(loc="upper right", fontsize=14)
plt.xlabel("Epoch", fontsize=14)
plt.ylabel("RMSE", fontsize=14)
plt.show()

print('OK')
```

```
Training...n_epochs= 500
```



```
OK
```

Qd Explain the Polynomial RSME-Capacity plot

Now we revisit the concepts from `capacity_under_overfitting.ipynb` notebook and the polynomial fitting with a given capacity (polynomial degree).

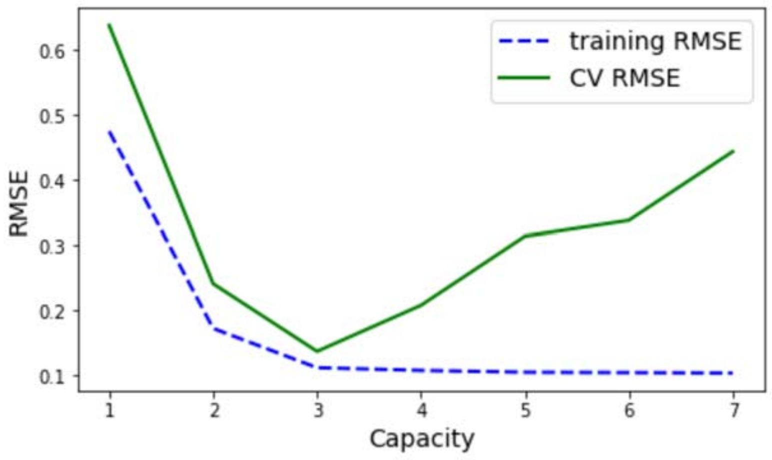
Peek into the cell below, and explain the generated RSME-Capacity plot. Why does the *training error keep dropping*, while the *CV-error drops* until around capacity 3, and then begin to rise again? The Cross validation RMSE begins to rise again after capacity 3 as the model is starting to be overfitted thus loosing its predictive performance, while the training RMSE keeps getting lower as we saw earlier in Qa

What does the axis Capacity and RMSE represent? The Capacity represent the polynomial degrees, and the RMSE represents the root mean squared error

Try increasing the model capacity. What happens when you do plots for `degrees` larger than around 10? When doing degrees larger than arorund 10, the CV RMSE just sky rockets exponentially, from 0.60 at 8 to 38.94 at 10 and 154.97 at 11.

```
In [19]: # TODO: Qd...code review

         # NOTE: modified code from [GITHOML], 04_training_linear_models.ipynb

         %matplotlib inline

         from math import sqrt
         import numpy as np
         import matplotlib.pyplot as plt

         from sklearn.pipeline import Pipeline
         from sklearn.preprocessing import PolynomialFeatures
         from sklearn.linear_model import LinearRegression
         from sklearn.model_selection import cross_val_score
         from sklearn.metrics import mean_squared_error

         def true_fun(X):
             return np.cos(1.5 * np.pi * X)

         def GenerateData():
             n_samples = 30
             #degrees = [1, 4, 15]
             degrees = range(1,8)

             X = np.sort(np.random.rand(n_samples))
             y = true_fun(X) + np.random.randn(n_samples) * 0.1
             return X, y, degrees

         np.random.seed(0)
         X, y, degrees  = GenerateData()

         print("Iterating...degrees=",degrees)
         capacities, rmses_training, rmses_cv= [], [], []
         for i in range(len(degrees)):
             d=degrees[i]

             polynomial_features = PolynomialFeatures(degree=d, include_bias=False)

             linear_regression = LinearRegression()
             pipeline = Pipeline([
                     ("polynomial_features", polynomial_features),
                     ("linear_regression", linear_regression)
                 ])

             Z = X[:, np.newaxis]
             pipeline.fit(Z, y)

             p = pipeline.predict(Z)
             train_rms = mean_squared_error(y,p)

             # Evaluate the models using crossvalidation
             scores = cross_val_score(pipeline, Z, y, scoring="neg_mean_squared_error", cv=1
         0)
             score_mean = -scores.mean()

             rmse_training=sqrt(train_rms)
             rmse_cv=sqrt(score_mean)

             print(f"  degree={d:4d}, rmse_training={rmse_training:4.2f}, rmse_cv={rmse_cv:4
         .2f}")

             capacities    .append(d)
             rmses_training.append(rmse_training)
             rmses_cv      .append(rmse_cv)
```

```
Iterating...degrees= range(1, 8)
  degree=   1, rmse_training=0.48, rmse_cv=0.64
  degree=   2, rmse_training=0.17, rmse_cv=0.24
  degree=   3, rmse_training=0.11, rmse_cv=0.14
  degree=   4, rmse_training=0.11, rmse_cv=0.21
  degree=   5, rmse_training=0.10, rmse_cv=0.31
  degree=   6, rmse_training=0.10, rmse_cv=0.34
  degree=   7, rmse_training=0.10, rmse_cv=0.44
```



```
OK
```