```
# create the model
def model_fn():
    model = tf.keras.Sequential()
    # Adds a densely-connected layer with 256 units to the model:
    model.add(tf.keras.layers.Dense(256, activation='relu', input_dim=784))
    # Add Dropout layer
    model.add(tf.keras.layers.Dropout(rate=0.2))
    # Add another densely-connected layer with 64 units:
    model.add(tf.keras.layers.Dense(64, activation='relu'))
    # Add a softmax layer with 10 output units:
    model.add(tf.keras.layers.Dense(10, activation='softmax'))

    # compile the model
    model.compile(optimizer=tf.train.AdamOptimizer(0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

Data Augmentation

Data augmentation is a method for artificially generating more training data points. This technique is precipitated on the observation that for an increasingly large training dataset mitigates the problem of overfitting. For some problems, it may be easy to artificially generate fake data, while for others it may not readily be the case. A classic example where we can use data augmentation is in the case of image classification. Here artificial images can easily be created by rotating or scaling the original images to create more variations of the dataset for a particular image class.

Noise Injection

The noise injection regularization method adds some Gaussian noise to the network inputs during training. Also, Gaussian noise can be added to the hidden units to mitigate overfitting. Yet still another form of injecting noise into the network is to add some Gaussian noise to the network weights. Noise injection can be considered as a form of data augmentation. The amount of noise added is a configurable hyper-parameter.

Too little noise has no effect, whereas too much noise makes the mapping function too challenging to learn.

In TensorFlow 2.0, noise injection can be added to the model as a form of data augmentation using the method **'tf.keras.layers.GaussianNoise()'**. The **'stddev'** parameter of the method controls the standard deviation of the noise distribution. The following code listing shows an MLP Keras model with Gaussian noise applied to the model.

```
# create the model
def model_fn():
    model = tf.keras.Sequential()
    # Adds a densely-connected layer with 256 units to the model:
    model.add(tf.keras.layers.Dense(256, activation='relu', input_dim=784))
    # Add Gaussian Noise
    model.add(tf.keras.layers.GaussianNoise(stddev=1.0))
    # Add another densely-connected layer with 64 units:
    model.add(tf.keras.layers.Dense(64, activation='relu'))
    # Add a softmax layer with 10 output units:
    model.add(tf.keras.layers.Dense(10, activation='softmax'))

    # compile the model
    model.compile(optimizer=tf.keras.optimizers.RMSprop(),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

Early Stopping

Early stopping involves storing the model parameters each time there is an improvement in the loss (or error) estimate on the validation dataset. At the end of the training phase, the stored model parameters are used rather than the last known parameter before termination.

The technique of early stopping is based on the observation that for a sufficiently complex classifier, as the training phase progresses, the error estimate on the training data continues to decrease, whereas the validation data will see an increase in the model error measure. This is illustrated in Figure 34-2.