of features, which then grow to increasingly complex features as information flows to deeper layers of the network, to capture the mapping between the inputs and the target. See Figure 31-2.

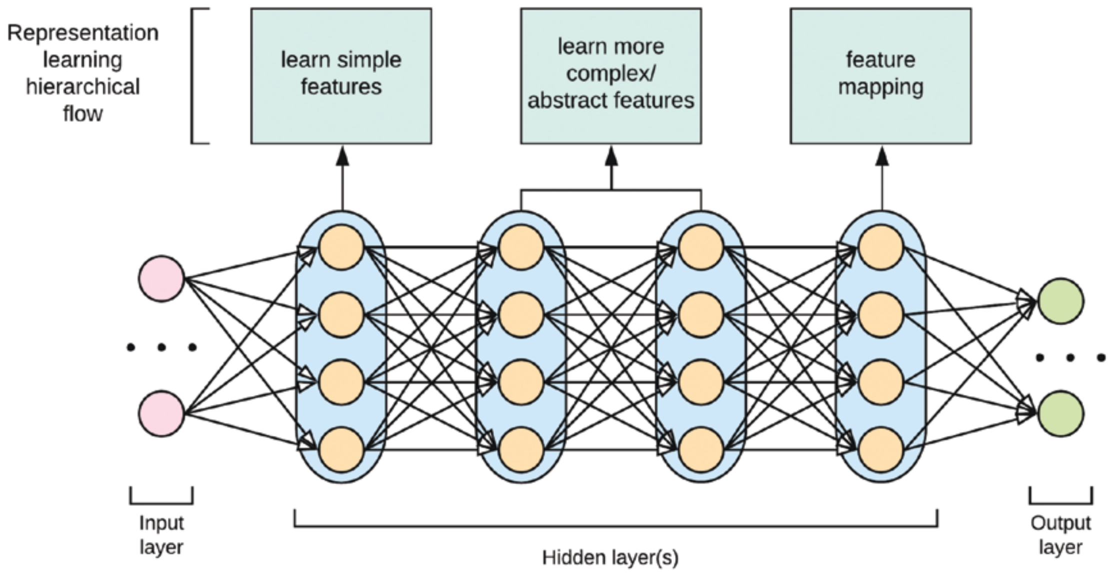
Representation learning hierarchical flow
learn simple features
learn more complex/ abstract features
feature mapping
Input layer
Hidden layer(s)
Output layer

Figure 31-2. *Hierarchical learning*

Choosing the Number of Hidden Layers: Bias/Variance Trade-Off

From experience, increasing the number of hidden layers may improve the representational quality of the network; however, arbitrarily increasing the number of hidden layers in your network design can have detrimental effects on the overall network performance with respect to generalizing to unseen observations. This is because the neural network will learn more closely the irreducible errors inherent in the training dataset and will fail to generalize to new examples.

Appropriate caution should be taken when selecting the number of hidden layers to avoid overfitting. Regularization techniques for neural networks such as Tikhonov regularization, Dropout, or early stopping are different methods of mitigating overfitting. Regularization for neural networks will be covered in more detail in a later section.

Empirically, one hidden layer will produce good results for simple learning problems, but if the number of output classes increases or there exists a high degree of non-linearities among the data features, then it is recommended to add more layers while taking care to ensure that the model performs well on test data. Choosing the number of neurons in a hidden layer and the number of hidden layers is usually a case of a trial-and-error heuristics and presents the case of applying hyper-parameter tuning to improve the network performance. Using a grid search for hyper-parameter tuning is a good way to approximate an optimal neural network architecture that performs well on test data.

Multilayer Perceptron (MLP) with Keras

In this section, we examine a motivating example by building an MLP model with Keras. In doing so, we'll go through the following steps:

- Import and transform the dataset.
- Build and compile the model.
- Train the data using **'Model.fit()'**.
- Evaluate the model using **'Model.evaluate()'**.
- Predict on unseen data using **'Model.predict()'**.

The dataset used for this example is the Fashion-MNIST database of fashion articles. This dataset contains 60,000 28 x 28 pixel grayscale images of ten clothing items (the target classes). This dataset is downloaded from the **'tf.keras.datasets'** package. The following code example will build a simple MLP neural network for the computer to classify an image of a clothing item into its appropriate class. The network architecture has the following layers:

- A dense hidden layer with 250 neurons
- A second hidden layer with 64 neurons
- A third hidden layer with 32 neurons
- An output layer with 10 output classes

```
# install tensorflow 2.0
!pip install -q tensorflow==2.0.0-beta0

# import packages
```