CHAPTER 34

Regularization for Deep Learning

Regularization is a technique for reducing the variance in the validation set, thus preventing the model from overfitting during training. In doing so, the model can better generalize to new examples. When training deep neural networks, a couple of strategies exist for use as a regularizer.

Dropout

Dropout is a regularization technique that prevents a deep neural network from overfitting by randomly discarding a number of neurons at every layer during training. In doing so, the neural network is not overly dominated by any one feature as it only makes use of a subset of neurons in each layer during training. In doing so, Dropout resembles an ensemble of neural networks as a similar but distinct neural network is trained at each layer. Dropout works by designating a probability that a neuron will be dropped in a layer. This probability value is called the Dropout rate. Figure 34-1 shows an example of a network with and without Dropout.

E. Bisong, *Building Machine Learning and Deep Learning Models on Google Cloud Platform*,
https://doi.org/10.1007/978-1-4842-4470-8_34

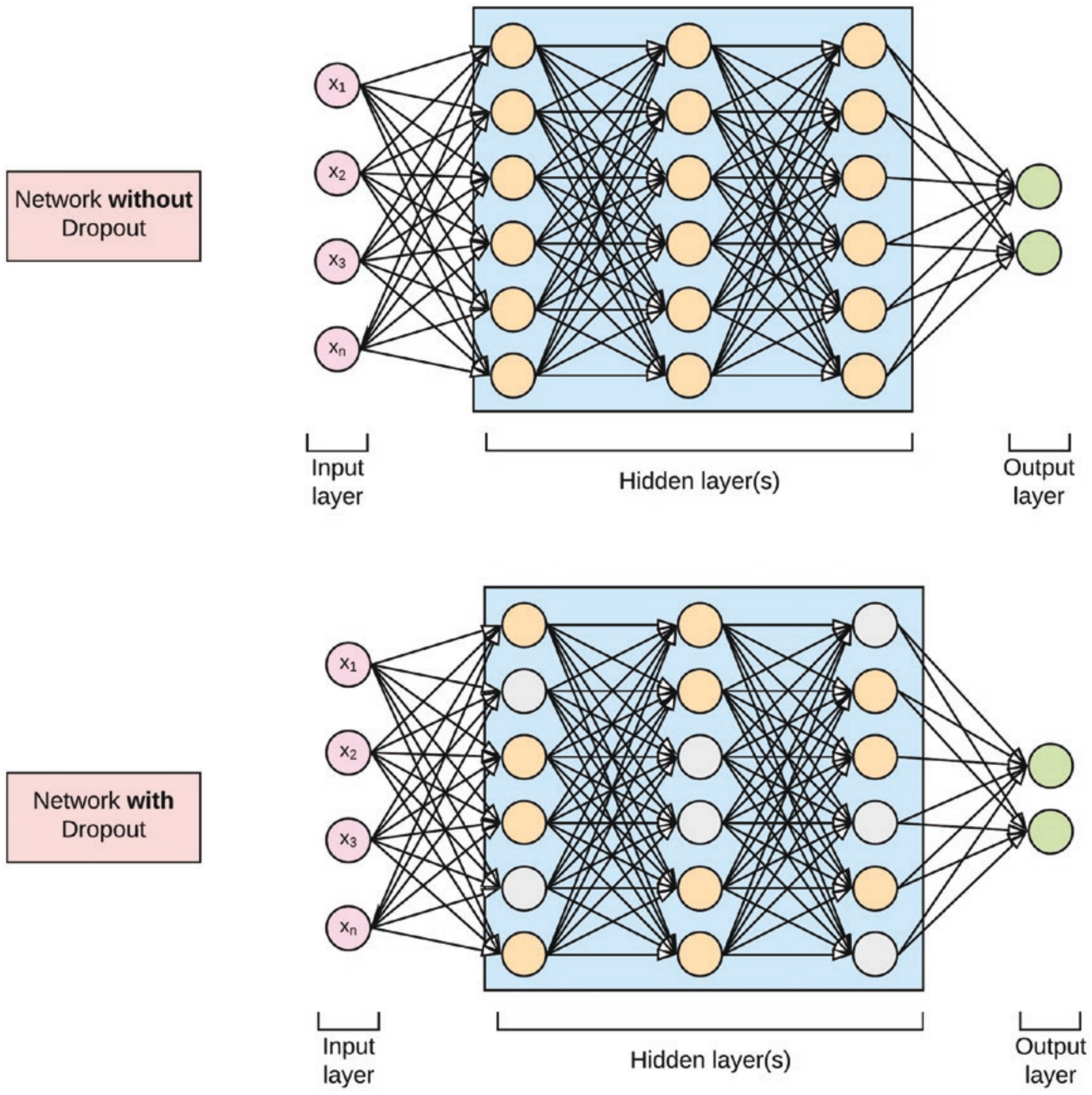

Figure 34-1. *Dropout. Top: Neural network without Dropout. Bottom: Neural network with Dropout.*

In TensorFlow 2.0 Dropout is added to a model with the method **'tf.keras.layers.Dropout()'**. The **'rate'** parameter of the method controls the fraction of the input units to drop. It is assigned a float value between 0 and 1. The following code listing shows an MLP Keras model with Dropout applied.

```
# 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.