Article from

Predictive Analytics and Futurism

December 2019

Autoencoders for Anomaly Detection

By Jeff Heaton

In data science, anomaly detection is the identification of unusual items, events or observations that raise suspicions by differing significantly from previously seen data. Typically, the anomalous items will translate to some kind of problem such as bank fraud, a structural defect, medical problems or errors in a text. Anomalies are also referred to as outliers, novelties, noise, deviations and exceptions. Anomaly detection can also be particularly useful to determine how suited a model trained on a particular dataset is at handling a new dataset. This suitability detection is the focus of this article.

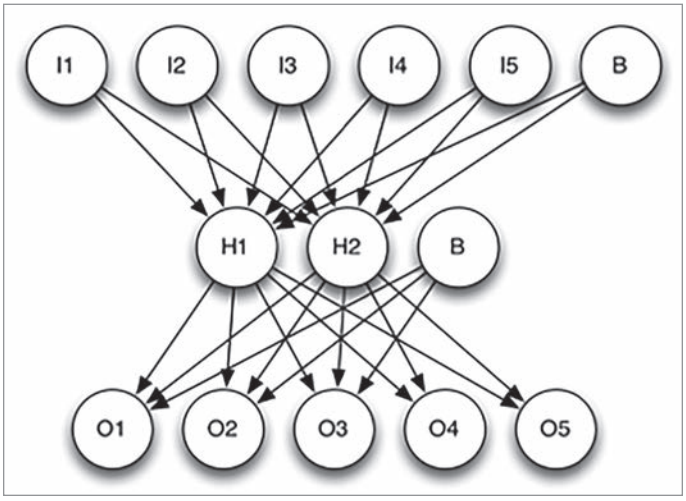
INTRODUCTION TO AUTOENCODERS

An autoencoder is a type of neural network that has the same number of input neurons as output neurons. The number of input/output neurons you have corresponds to the size of your feature vector after the data source has been encoded. For instance, you might have a single input for continuous and a set of dummy variables for each of your categorical inputs. The autoencoder is trained in a supervised fashion; however, the x (inputs) and y (targets) are the same. It is also important to note that the autoencoder is using the ability of a neural network to perform a multi-output regression. The neural network is learning to directly copy the inputs to the outputs. This structure is seen in Figure 1.

At first glance the autoencoder may not seem that useful. We are training a neural network to simply pass the input through to the output. However, there is always at least one hidden layer with fewer neurons than the input and output layers. These hidden layers teach the neural network to compress the input data. You can think of the connections between the input and hidden layers as learning to be a data decompressor and the connections between the hidden and output layers as learning to be a decompressor. It is common to separate the autoencoder into two neural networks. This way the hidden layer becomes the output layer for the compressor.

Extracting the output from the hidden layer can be thought of as a form of dimension reduction, similar to principal component analysis (PCA) or t-distributed Stochastic Neighbor Embedding (t-SNE). Because the neural network shown in Figure 1 contains only two hidden neurons, it would reduce the dimensions from the five input neurons down to two dimensions.

Figure 1
Autoencoder Structure



Despite the fact that an autoencoder is trained like a normal supervised neural network, usually using some variant of back-propagation, this training process is considered unsupervised. This is because no one value from the dataset is the target—all of the values from the dataset are the target value. In this regard, the autoencoder training is unsupervised in the same sense as PCA or t-SNE are not provided with a target. However, unlike PCA or t-SNE, the autoencoding neural network also includes a decoder. T-SNE and PCA both lack a well-defined means of returning to the high-dimension input that they processed. In this regard, an autoencoder shares more with a compression algorithm, such as PKZIP, than a dimensionality reduction algorithm.

You can essentially think of the training process of the autoencoder as creating a compression algorithm optimized to the data you provided. Such domain-specific compression-decompression (codec) algorithms are not uncommon. Portable Net Graphics (PNG) format is a lossless codec for image compression. The Joint Photographic Experts Group (JPEG) format is a lossy codec for image compression. A lossy codec will lose some of the original detail from the source data; a lossless codec maintains absolute data integrity. For images and audio, absolute data integrity is not always required.

This specialization among codecs is what allows an autoencoder to be used for anomaly detection. Early cellphone compression algorithms were designed to compress human voice as effectively as possible and make the best utilization of the very slow

cellular networks. When non-voice sounds, such as music, were compressed with these early voice-centric codecs, the music would clearly sound distorted. These early cell phones were anomaly detectors. They produced very little distortion among the human voice data that they were designed for and very high distortion on all other sounds. The more noise introduced into the signal, the less similar that signal was to the original type of data the codec was designed for. Essentially, the effectiveness of the specialized lossy codec for a particular dataset shows how much of an anomaly the new dataset is when compared to the original dataset the codec was designed for.

Now consider an autoencoder. We create an anomaly detector by training this autoencoder on data that we consider "normal." Overfitting is not that big of a concern, since this is effectively an unsupervised learning; however, a k-fold or similar scheme might be used for early stopping of the neural network training once the out of sample error ceases to improve.

INTRODUCTION TO THE KDD-99 DATASET

The KDD-99 dataset is famous in the security field and almost a "hello world" of intrusion detection systems in machine learning. This dataset was used for the Third International Knowledge Discovery and Data Mining Tools Competition, held in conjunction with the Fifth International Conference on Knowledge Discovery and Data Mining. According to the KDD archive, "The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between 'bad' connections, called intrusions or attacks, and 'good' normal connections. This database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment."[1] This dataset is commonly used for computer security and anomaly detection examples. This is the dataset that I used for this example on autoencoder anomaly detection.

The KDD-99 dataset includes a target that identifies the type of attack or if the transaction was normal. We will not directly use this target in the training. Rather, we will separate the data into normal and attack rows. We will train the neural network on the normal rows. We will then compare the difference between the root mean square error (RMSE) for normal vs. error. This RMSE is the difference between the data before and after the autoencoder compresses and decompresses it. The RMSE effectively measures the amount of noise added by running through the autoencoder. Just to be sure there is no overfitting, we will compare out-of-sample normal to the error rate for normal as well.

ANOMALY DETECTION EXAMPLE

This example is from a college course that I teach on deep learning. I will not reproduce all of the code here.[2] This example is in the Python programming language, using TensorFlow 2.0 for deep-learning support.

The TensorFlow autoencoder neural network is set up by the following lines of code:

```
model = Sequential()
model.add(Dense(25, input_dim=x_normal.shape[1],
  activation='relu'))
model.add(Dense(3, activation='relu'))
model.add(Dense(25, activation='relu'))
model.add(Dense(x_normal.shape[1])) # Multiple
  output neurons
model.compile(loss='mean_squared_error',
  optimizer='adam')
model.fit(x_normal_train,x_normal_
  train,verbose=1,epochs=100)
```

You can see that the number of input neurons and output neurons are the same, specified by the value x_normal.shape[1]. These both correspond to the number of predictors in the feature vector generated from the KDD-99 dataset. There are additionally 25 neurons added before and after the three bottleneck neurons to assist with compression and decompression. The three hidden layer neurons specify the number of dimensions that the autoencoder is reducing the data to.

The results from the experiment are shown below.

```
In-Sample Normal Score (RMSE): 0.30
Out of Sample Normal Score (RMSE): 0.31
Attack Underway Score (RMSE): 0.53
```

The in-sample and out-of-sample normal data RMSE were approximately the same, between 0.30 and 0.31. The attack rows were noticeably higher at an RMSE of 0.53. This is consistent with anomaly detection in that the anomaly data is compressed with more noise than normal data. ■

Jeff Heaton, Ph.D., is vice president and data scientist at RGA Reinsurance Company, Inc. He can be reached at *jheaton@rgare.com*.

ENDNOTES

1 University of California, Irvine. KDD Cup 99 Data: Abstract. *KDD Archive*. Last modified, Oct. 28, 1999. *http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html*.

2 For the complete Python source code for the example, see Heaton, Jeff. T81-588: Applications of Deep Neural Networks. *GitHub*. Last updated, Aug. 20, 2019. *https://github.com/jeffheaton/t81_558_deep_learning/blob/master/t81_558_class_14_03_anomaly.ipynb*.