Article from

Predictive Analytics & Futurism

December 2018

Issue 19

The Possible Role of Convolutional Neural Networks in Mortality Risk Prediction

By Holden Duncan

How might a computer tell the difference between the road and a tree while steering a car at 40 miles per hour? A rules system for each possible object that might be encountered could be created; however, such a system only allows for a limited amount of features, and is only as effective as the rules themselves. Increasing the number of rules might make for a more accurate classification, but such an engine would become unmanageable. In addition, attempting to hand-engineer such features limits a model to human intuition of pixel-by-pixel photo recognition.

There are already articles about random forests or linear/logistic regressions, but these methods involve the use of functions and hand-engineered features composed of different categories. As such, these models are generally unable to identify wholly new ideas without specific encoding. However, a model that excels in using the spatial relationship between complex features in data to generate accurate classification could learn to recognize new patterns. A convolutional neural network, or CNN, learns to use concrete low-level features in order to extract identifying abstract ideas. The nodes or "neurons" of the network are organized into different interconnected layers and through training becomes a predictive engine similar to the human brain. While CNNs are most commonly used in image classification, I hope to instead apply them to model mortality risk through visual representations of medical history.

WHAT IS A CONVOLUTIONAL NEURAL NETWORK

Convolutional neural networks are specialized neural networks. Regular neural networks take some fixed-shape input and produce an output. The input propagates through the network via the weighted connections between different layers of nodes, and the transformations which take place in and between nodes produce optimized predictions. Assuming layer A is the layer just before layer B in a given network then:

- Each node in layer A has a weighted connection to every node in layer B.
- Let A_x represent a given node in layer A, similarly for B_y in B, and let $Weight_{xy}$ be the weight of the connection between those two nodes. Then the input from A_x to B_y may be expressed as:

$$Ax_{out} * Weight_{xy}$$

- The net input to node B_y is the sum of the inputs from each node in A to B_y:

$$By_{net} = \sum_{x\ in\ A} Ax_{out} * Weight_{xy}$$

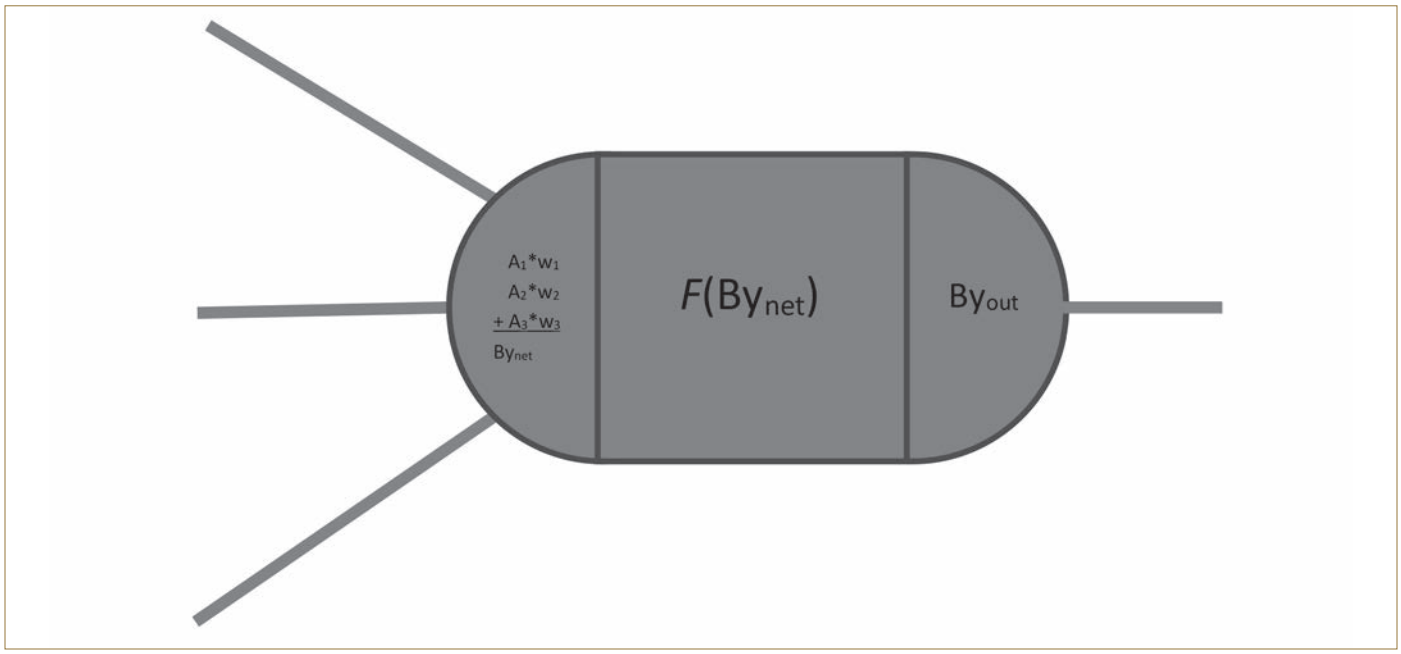
- The output of node By is determined by the result of the activation function, F, applied to the net input:

$$By_{out} = F(By_{net})$$

This process is shown visually in Figure 1 (pg. 21).

Generally, a neural network consists of an input layer and an output layer with any number of hidden layers in between. Each layer may contain any number of nodes. The model makes

Figure 1
Node Output Depends Upon Net Input and Activation Function



accurate predictions using the weight associated with each connection and is able to learn by optimizing these weights. When a fresh network is initialized, these weights are randomly assigned small nonzero values.

Unfortunately, we are unable to hand-wave meaningful weights into existence, but thankfully the network learns through a variant of trial and error and not human intuition. Training occurs through backpropagation, a form of supervised training which compares the network generated output with the expected output by using labeled data, e.g., data labeled "Duck" would have an expected output of 1.0 for "Is Duck" and 0.0 in any other category. In essence, the training data is passed forward through the network and the error is found, and then the network works backwards making adjustments. The Mean Squared Error is commonly used to measure error, thus the total error, Etotal, for the output layer may be expressed as:

$$\sum_{\substack{Node\ in \\ Layer}} \frac{1}{2}(target - output)^2$$

An important feature of backpropagation is that changes are applied to each weight individually. The weight of each connection is increased or decreased by the partial derivative of the total error, E_{total}, with respect to the given weight wx. Using the chain rule the error associated with a given weight may be expressed as:

$$\frac{\partial E_{total}}{\partial w_x} = \frac{\partial Etotal}{\partial O_{1_{out}}} * \frac{\partial O_{1_{out}}}{\partial O_{1_{net}}} * \frac{\partial O_{1_{net}}}{\partial w_x}$$

Because E_{total} represents the amount of change needed to reduce the error to zero, the partial derivative with respect to a given weight yields the amount by which that weight must change to minimize the total error. This change is often multiplied by some learning rate to make training more efficient. (These examples assume a learning rate of 1 for simplicity.)

Convolutional neural networks go a step further and use specialized convolution and pooling layers. The output of these layers may be a two-dimensional matrix, or a matrix of matrixes called a tensor. Convolution layers and pooling layers both have kernel and stride dimensions. These variables remain constant within a layer, but may change between layers. The shape of the kernel acts like a spotlight and highlights a limited region of the total input. The region is processed and then the kernel is translated across the input according to the stride.

In convolution layers, the highlighted region is multiplied by a filter. The filter may be matrix or tensor. The sum of the elements in the product represents the similarity between a given region and the filter. The resulting activation map not only shows whether the filter was activated, but also where in the input that filter was activated. There may be any number of filters in a given layer. Multiple filters in a layer will produce a tensor of the various activation maps "stacked" one after another. In early layers these filters detect simple features from concrete

input values. In later layers however, filters tend to represent more abstract ideas using the spatial relations between earlier filters. The network operates much like how a human identifies a high-level idea, like a square versus a rectangle, by the low-level information such as the relative positions of each edge.

Pooling layers, on the other hand, serve to only down sample input. A typical method is max pooling, during which the largest value from the input region becomes a single value in a smaller matrix. The result is similar to the input except the location of details are more generalized. Pooling layers are useful because not only do they reduce the dimensions of the matrices within the model, but these layers also help prevent overfitting.

An overfit model begins to simply memorize strict patterns found in training examples. A network trained to detect cars might overfit and expect any detected wheels to be perfectly horizontal. Pooling the layer that detects wheels retains any spatial relationships, but is less sensitive to the exact locations of the features. This generalization makes for a more flexible model.

APPLYING CONVOLUTION NEURAL NETWORKS TO UNDERWRITING DATA

CNNs are not limited to computer vision, however. Because of how these networks use different low-level features to extrapolate complex and abstract relationships, such networks may be used beyond classical images. While the ability to recognize abstract ideas from spatially related features is commonly used for image classification, images are just organized tensors. As such, any tensor or matrix of spatially related data may be used as the input to a CNN.

I have generated visual representations of individuals' prescription histories. The x-axis represents time and the y-axis represents how dangerous the prescription is considered. Darker sections represent a higher number of prescriptions of that severity filled within a given time interval. I have chosen this approach because a convolutional model trained this way may be able to find and use unknown patterns. A note of caution, however: Because most data in columns may be shuffled vertically without the loss of information, tabular data tends not to be a good candidate for a CNN. If one could shuffle the columns of an image and not scramble the picture, then the spatial relationships would be insignificant. The prescription histories, like a picture, have an inherent ordering to the columns. Thus, they can benefit from a CNN analysis.

CONCLUSIONS AND FUTURE DIRECTIONS

The output of a convolution layer with more than one filter is a matrix of matrices. This resulting tensor is passed through the network in order to generate some target output. In order to build, train and test different models, I used Google's TensorFlow library for Python as the backend for the Keras module by François Chollet. The logic for the actual training and creation of the model is based in C and C++ with Google's Python wrapper for interaction. The Keras package is then used to implement TensorFlow as Keras has friendlier syntax and added tools for data manipulation.

Moving forward I plan to use major drug groups, sub groups or even active ingredients in place of severity scoring, thereby possibly capturing new relationships between medications. Theoretically, this network could even diagnose patients through symptom history. In addition, the output may be more than a yes or no answer, but instead a vector predicting the mortality risk of the individual for each coming year. And while I have generated actual images, any two- or three-dimensional input of spatially related data could be used. Such technology is only limited by human creativity and available data. ■

Holden Duncan is a data scientist at RGA Reinsurance Company, in Chesterfield, Mo. He can be reached at *HoldenDDuncan@gmail.com*