Week 2: Neural Networks and Business Processes

Nate Bachmeier

TIM-8150: Artificial Intelligence

October 11th, 2020

Northcentral University

# Neural Networks and Business Processes

The human brain is a supercomputer, with capabilities to reason about situations even when facing ambiguity and imperfect information. In 1949, neural scientists found that the mechanism that accomplishes this feat is a mesh of connected *neurons* (Lukac, Milic, & Nikolic, 2018). Despite unlocking the biological key to producing cognitive learning, the processing power necessary to digitally create these structures was not widely available until the early 2000s. Now cloud technologies, such as Machine Learning as a Service (MLaaS) and Hardware Acceleration as a Service (AaaS), enable ubiquitous access to these data structures for enhancing decision processes. While most organizations understand the value proposition of leveraging technologies like MLaaS, it can be challenging to get started (Garbuio & Lin, 2019). These barriers to entry come with questions around (1) what scenarios are most applicable, (2) how these systems work, and (3) what are some considerations along the way.

## Application to Business Problems

A traditional software program combines *data* and *rules* to produce *outputs*. In contrast, machine learning software takes *data* and *outputs* to derive *rules*. This derivation comes from four major strategies: association rule mining, regression modeling, classification techniques, and clustering techniques (Barua & Mondal, 2019). Many business challenges fit into these categories, such as supply chain management (SCM). SCM systems need to forecast inventory levels through regression analysis. Next, it must classify vendors and choose the right levels to source from each one. As the inventory arrives at various outlets, their performance metrics enable systematic administration by reviewing cluster associations. Finally, when customers purchase a pair of pants, association rules can suggest a matching shirt. Countless additional business scenarios exist with fuzzy rules or general uncertainty.

## Evolution of the problem

Numerous organizations begin their journey into intelligent systems with statistical modeling and variance analysis. These approaches work for many linear models but tend to break down for non-parametric functions (Waal & Toit, 2011). For example, a business wants to appraise houses given a collection of features about the home. Houses come in all shapes and sizes, and this makes it challenging to compare those features directly. Instead, the appraiser must approximate a function that considers these characteristics and their weighted importance. Meanwhile, another company needs to classify handwritten digits, which requires mapping a 32x32 pixel image as its numeric value. Both scenarios and countless more require a mechanism to translate these non-parametric functions into parametric approximations.

## Nature’s solution

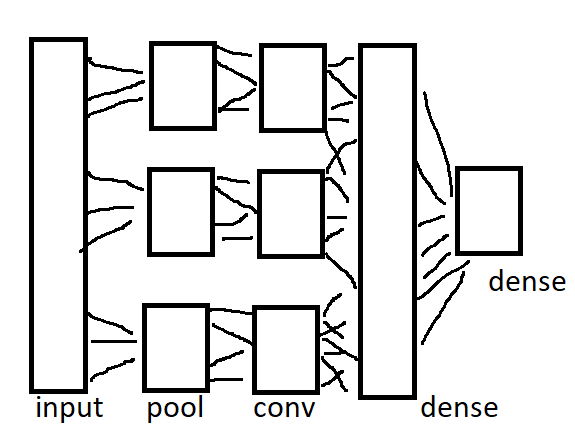
In biology, animal brains accomplish these tasks through meshes of neurons that transmit signals across connected synaptic (transforming) and activation (filtering) links (Keller, Liu, & Fogel, 2016). Later, that animal sees an object, and its brain encodes the image into a feature map. These features traverse the neural network and output a collection of responses, such as its food and ten feet away. Over time, the creature *learns* if those responses are correct and revise network weights to encourage or avoid similar situations. Data scientists and mathematicians replicate these ideas by calibrating edge weights, called backpropagation, on connected graphs called neural networks.

## Network Structure

A neural network consists of three building blocks called the input, hidden, and output layers. For instance, an animal image classification system might assign 64x64 pixel images into ten predetermined categories. This example requires an input layer with 4096 neurons, an output layer of ten neurons, and some hidden layers in the middle. Adding more hidden layers enables extracting more details from the image like object edges (layer-1), ears (layer-2), cat’s ears (layer-3), to a tiger’s ears (layer-4) (Fridman, 2017). While more complex networks can extract more insights, it comes with a cost of needing exponentially more data.

Depending on the connectivity configuration, this can become too expensive and require model compression strategies (Cheng, Wang, Zhou, & Zhang, 2018). For instance, the input layer could feed into a series of pooling transforms that downgrade the resolution by averaging every 2x2 pixels. Another tactic might focus on connecting and evaluating local segments of neurons before outputting into global join constructs for prediction (see Figure 1). Meanwhile, other situations like the housing model perform better fully connected to a shallow pipeline. While standard architectures exist for many problem areas, some experimentation is necessary.

Figure 1: Network Structure



# Experimenting with Neural Networks

Building an image classification system with Keras and Tensorflow provides an efficient framework for neural network experimentation and parameter manipulation.

## Data Collection

Yelp.com assists hungry patrons in finding new and exciting restaurants by collects pictures and business reviews from users. A subset of their knowledge graph is anonymized and publically shared for the research community (Yelp.com, 2020). This export contains 160,000 (5.4 GiB) high-resolution JPEG images and associated metadata, such as its category: food, drink, indoor, outdoor, or menu.

## Preparing and Loading

Keras’ image\_dataset\_from\_directory is a utility method for loading batches of reading images and uses the folder name as the label. Using this function requires the files to be sorted in advance (see Figure 2). The sort operation also produces a normalized grey-scale duplicate for experimenting with different input sizes. After completing these operations, the content was lazy-loaded into an Amazon SageMaker ml.c5.9xlarge Notebook (see Figure 3). Despite the instance having 72GiB of memory, the notebook crashes if the images exceed 128x128 pixels. However, the accuracy of 64x64 was comparable and used 75% less memory. Another challenge came from the utility method ships with TF 2.4-prerelease and is installable via the command pip install tf-nightly.

|  |  |
| --- | --- |
| Figure 2: Preprocessing Files | Figure 3: Load Content |

## Training Model

Tensorflow’s tutorials include an image classification example with different types of flowers (Tensorflow, 2020). Their vanilla model (see Figure 4) begins with (1) an input layer equal to the image batch shape, (2) then feeding this into a series of 2-D convolutional and max-pooling. This approach partitions the graphic into sections and locally reduces each unit. Next, the final output of the Conv2D steps is (3) fully connected with a join-combiner construct. This layer (4) connects into a dense layer with one neuron per class.

|  |  |
| --- | --- |
| Figure 4: Train Model | Figure 5: Results |

## Reducing overfitting

Suppose the training accuracy is nearly perfect, and the validation line makes a cup. In that case, it could be evidence of overfitting (see Figure 5). Models that are overfitted or under fitted are not generalizable and do not perform well in production environments. One solution is to apply a drop-out filter to randomly remove activation signals between layers (Srivastava, Hinton, Krizhevsky, & Sutskever, 2014). Another approach is to manipulate the training input to include transformations like zooming, cropping, and rotations (see Figure 6).

After making these changes, the training and validation curve more closely overlap, providing greater confidence that the model will be around 86% accurate in production (see Figure 7). These results are reasonable, given only ten iterations (epochs) of training time. Using a 34 core cloud instance still takes 6.5 minutes, highlighting the need for hardware acceleration, such as General Purpose Graphic Processing Units (GPGPUs).

|  |  |
| --- | --- |
| Figure 6: Remove overfitting | Figure 7: Results |