**Deep Learning with R – Chapter 1**

* The central problem in machine learning is to *meaningfully* transform data: how can we provide the computer with meaningful representations.
* *Learning,* in the context of machine learning, describes an automatic search process for better representations.
* Deep learning is a specific subfield of machine learning that puts an emphasis on learning successive layers of increasingly meaningful representations.
  + The number of layers contributing to a model of the data is called the *depth* of the model.
* Deep learning is, technically, a multistage way to learn data representations.
* What a layer does to its input is determined by its weights, parameters that must be learned through exposure to the data.
  + Learning means finding the set of values of all layers in a network, such that network will correctly map example inputs to their associated targets.
* To control the output of a neural network, you need to be able to measure how far this output is from what you expected. This is the job of the *loss function* of the network.
  + The loss function takes the predictions of the network and the true target and compute a distance score, capturing how well the network has performed.
  + The value of the loss function is then used to adjust the weights of the network in the correct direction.
* Together with performance gains, deep learning also automates what used to be the most crucial step in a machine learning workflow: feature engineering.

**Deep Learning with R – Chapter 2**

* In machine learning, a category in a classification problem is called a *class*. Data points are called *samples*. The class associated with specific samples is called a label.
* Basic machine learning workflow:
  + We feed the neural network the training data.
  + The network will then learn to associate images and labels.
  + We ask the network to produce predictions for the test data and we then verify whether these predictions match the labels from test labels.
* The core building of a neural network is the layer, a data-processing module that you can think of as a filter for the data. Some data goes in, and it comes out in a more useful form.
  + Layers extract *representations* of the data that is fed into them.
* Tensors are a generalization of vectors and matrices to an arbitrary number of dimensions.
  + A scalar is a tensor that contains only one number.
  + A one-dimensional array of numbers is called a vector, or 1D tensor.
  + A matrix is a two-dimensional array of numbers (a 2D tensor). A matrix has two axes (often referred to as *rows* and *columns*).
    - If you pack such matrices in a new array, you obtain a 3D tensor, which you can visually interpret as a cube of numbers.
  + A tensor is defined by three key attributes
    - Number of axes (rank): a 3D tensor has three axes, and a matrix ha two axes.
    - *Shape*: an integer vector that describes how many dimensions the tensor has along each axis.
    - *Type*: The type of data contained in the tensor; for instance, a tensor’s type could be *integer* or *double*.
  + In general, the first axis in all data tensors you’ll come across in deep learning will be the *samples* axis.
  + Deep-learning models don’t process an entire dataset at once; rather, they break the data into small batches.
* Vector data is the most common case of tensors. In such a dataset, each single data point can be encoded as vector. Thus, a batch of data will be encoded as a 2D tensor (an array of vectors), where the first axis is the *sample axis* and the second axis is the *feature axis*.
* Time or sequence data is stored in a 3D tensor:
  + The first axis is the sample axis, and can represent, for example, one trading day.
  + Each sample has more than one place in time. For each minute of a trading day, we could have 390 minutes of price data (this is the time axis).
  + For each sample in a time sequence, we might have up to *m* features representing characteristics of the sample at that particular moment.
* Image data is stored in a 4D tensor:
  + The first axis is the sample axes and represents each individual sample.
  + The next two axes represent height and width of each image.
  + The fourth axis represents the color depth of each sample.
  + All transformations learned by deep neural networks can be reduced to a handful of tensor operations applied to tensors of numeric data.
  + For instance, the **relu** operation is an element-wise operation: operations that are applied independently to each entry in the tensors being considered.
  + Operations involving tensors of different dimensions involve *sweeping* the operation across one of the dimensions of the largest tensor. For instance, to add a vector to a matrix, we could add the vector to all columns of the matrix or add the vector to all the rows of the matrix.
  + The dot operation, also called *tensor* *product* combines entries in the input tensors (a dot product, basically, is the projection of one vector over another). The larger a dot product, the smaller the angle between them (the larger the projection of one over the other).
  + Reshaping is another common tensor operation. Reshaping a tensor means rearranging its rows and columns to match a target shape. A special case of reshaping is transposing a matrix, so its rows become it columns and vice-versa.
  + All tensor operations have geometric interpretations.
  + Remember that applying a matrix to a vector means finding new coordinates for that vector in a space defined by the matrix.
  + Neural networks consist entirely of chains of tensor operations and that all of these tensor operations are just geometric transformations of the input data. It follows that you can interpret a neural network as a very complex geometric transformation in a high-dimensional space, implemented via a long-series of simple steps.
  + Deep learning provides a way to decompose a very complicated geometric transformation a long chain of elementary ones.
    - Each layer disentangles the data a little – and a deep stack of layers makes tractable an extremely complicated disentanglement process.
  + Each layer of a neural network contains attributes called *weights* or *trainable parameters*.
    - Initially, we assign random values to these parameters. Although the representation resulting from this random transformation will be meaningless, they will allow us to gradually adjust them.
    - We update the weights using the derivative of the loss function with respect to the weights.
      * The gradient of any function is a vector has the direction of the greatest increase of the function at a point *f* (x), while its magnitude is the rate of increase in that direction.
      * By moving in the opposite direction of the gradient, we are, in effect, minimizing *f* (x)
  + The optimal parameters, that is, the weights that minimize the mismatch the loss function could be found, theoretically, analytically. If the loss function is convex, then its minimum would be at the value that its derivative is equal to 0. However, this is intractable for humans after a certain point, because that would mean finding the partial derivatives of thousands of individual weights.
  + Stochastic gradient descent is a method that involves calculating the gradient of the loss function in a random batch of data and updating the weights based on the results.
  + Description gradient descent:
    - Draw a batch of training samples x and corresponding targets y
    - Run x through the network to obtain predictions (y\_pred)
    - Compute the loss of the network on the batch, i.e., a measure of mismatch between y\_pred and y.
    - Compute the gradient of the loss with regard to the network’s parameters (*a backward pass*)
    - The gradient is a vector and contains a value for each weight in W (the direction in that dimension). Add this new vector to your original weights so you “move then” in the opposite direction, therefore minimizing the loss.
    - When updating the weights, analysis usually multiply the gradient by a learning parameter (the *step factor*), which “smoothes” the changes in the weights, preventing them from:
      * Taking too long to converge (if the step is too small)
      * Or never converging.
  + Momentum is a concept utilized in some versions of gradient descent algorithms aimed at preventing the algorithm from getting stuck in a local minimum. Momentum updates the weights of a neural network based not only on the current value of the gradient, but also on the previous parameter update.
  + Since each neural network can be understood as a chain of transformations, we can calculate the derivative of the loss function with respect to each step. Based on the loss function, we first update the weights in the last layer and work backwards until the first.
* An iteration over all the training data is called an *epoch*
* Summary of two key concepts:
  + The loss is the quantity that you’ll attempt to minimize during training, so it should represent a measure of success for the task you are trying to solve.
  + The optimizer specifies the exact way in which the gradient of the loss will be used to update parameters.

**Deep Learning with R – Chapter 3**

* Neural networks workflow:
  + Layers map the input data to predictions
  + The loss function compares these predictions to the *targets*, producing a loss value
  + The optimizer uses this loss value to update the network’s weights.
* A layer is a **data-processing module** that takes as input one or more tensors and that outputs one or more tensors. Different layers are appropriate for different tensor formats.
* Building deep-learning models is done by clipping together compatible layers to perform useful data transformations.
* By choosing a network topology, the analyst effectively constrains what transformations are possible.
* After defining a network topology, it is necessary to define:
  + A loss function: the quantity that will be minimized during training.
  + An optimizer: an implementation of a specific variation of stochastic gradient descent.
* **Classifying movie reviews** **(Use case 1)**:
  + In the IMDB dataset, each review (sequence of words) has been turned into sequences of integers, where each integer stands for a specific word in a dictionary.
  + To put the reviews into a neural network, it is necessary to turn the reviews into tensors. There are two ways of doing this:
    - Pad the lists, so they have the same length, turn them into an integer tensor of shape (samples, word\_indices), and then use as the first layer in your network a layer capable of handling integer tensors.
    - One-hot encode the lists to turn them into vectors of 0s and 1s.
    - The input data is a vector: each review was transformed into a vector. The labels are scalers (1 and 0) representing review type.
    - A type of network that works well with this type of application is a simple stack of fully connected layers with relu activation.
      * *A relu activation means that each neuron will receive its inputs and apply a rectifier, defined as*
        + **.** The output is simple: if x is greater than 0, than the node will output the input value, otherwise it will output 0.
        + A fully connected layer means that every output is a function of every input.
    - In this application each layer will have 16 hidden units. This means that the weight matrix *W* will have 16 components.
      * Adding more hidden units might lead to overfitting data.
    - The question that every modeler face is about the number of hidden units to choose in each layer, and the number of layers itself.
  + In the absence of activation functions, the network would only be able to learn linear transformation of the input vector. By transforming the inputs using non-linear functions, we are able to explore a larger hypothesis space.
  + After deciding on the model, it is necessary to choose the loss function that the algorithm will have to minimize. For this application we will use binary cross-entropy.
    - Cross entropy loss measures the performance of a classification whose output is a probability between 0 and 1.
  + To monitor training accuracy, we will create a validation set by setting apart 10,000 samples from the original training data.
  + The model will be trained for 20 epochs (20 iterations over all samples in x\_train and y\_train tensors), in mini-batches of 512 samples:
    - From the training data, randomly draw 512 samples and update the weights until all training data has been used. Do that 20 times.
  + The loss plot shows that the loss is minimized around the fourth epoch. After that, each update to the weights does not improve accuracy on the validation set.
  + Example wrap-up:
    - Relu is a widely used activation function.
    - In a binary classification problem, the network should end with a dense layer with one unit and a *sigmoid* activation.
      * As a consequence, the loss function should be binary cross-entropy.
* **Classifying newswires (use case 2)**
  + This topic classification problem is similar to sentiment analysis, but there is an additional difficult: the number of output classes increased from 2 to 46.
    - To deal with this problem, we might want to create denser layers.
    - Small layers in complicated problems might function as information bottlenecks.
  + The activation function in the output layer is a *softmax* function. This function takes a vector of of length K, normalizes it into a probability distribution where each dimension corresponds to the probability of the sample belonging to that class.
  + The loss function that we are optimizing over is *categorical cross-entropy*. It measures the distance between two probability distributions: the vector outputted by the network and the true distribution of the labels.
  + Using layers that are significantly less dense might result in a loss of accuracy due to the fact that you will be trying to compress into an intermediate space that is too low-dimensional.
  + Extra: training a model with 128-hidden units:
    - The model’s accuracy peaked at the last epoch. However, since the fourth or fifth epoch, the accuracy/loss of the model stabilized. The improvement over the simpler model (64-hidden units was negligible).
* **Predicting housing prices (use case 3)**
  + Another common type of machine learning problem is *regression,* which consists of predicting a continuous value instead of a discrete label.
  + The dataset used in the example consists of the median price of homes in a Boston suburb in the mid-1970s.
  + Given that each feature is on a different scale, it is good practice to normalize it, so they have standard deviation of 1 and mean of 0. The values used for normalization should always come from the training data.
  + The network has no activation function in the output layer, because we do not want to constrain the range of the output.
  + The loss function is *mean squared error* (the square of the difference between the predictions and the target).
  + Given that the dataset is so small, leaving aside a single validation set to track the model would generate highly variable estimates. The way to mitigate this problem is using K-fold cross-validation:
    - K-fold cross-validation consists of splitting the available data into K partitions, instantiating K identical models, and training each one on K-1 partitions while evaluating on the remaining partition.
  + The model calculates the weights on 303 samples and uses that to predict the left-out sample. After the model goes through each sample 100 times (number of epochs), the trained weights will be used to predict the values in the validation set. This is done 4 times. Finally, the model is averaged over.
  + The model stops improving after 39 epochs. So, we then try a simpler model with 39 epochs.
  + When little training data is available, it is preferable to use a small network with few hidden layers in order to avoid severe overfitting.
* If your data is divided into many categories, you may cause information bottlenecks if you make the intermediate layers too small.