Coursera

Notes

Neural Networks and Deep Learning

Carlos García González

Summer 2020

Contents

Chapter 1

Neural Networks and Deep Learning

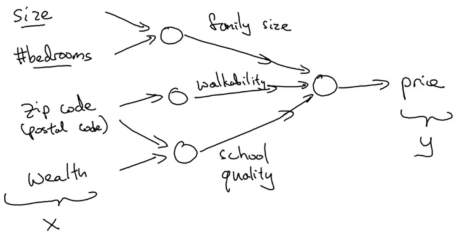
Week 1

What is a Neural Network?

The housing price prediction problem can be seen as the simplest of Neural Networks. First, lets start by assuming that the house pricing is only affected by its size.

Size x -¿ Neuron (does the determined function) -¿ Price y

In this particular case, a RELU (Rectified Linear Unit) function [1.1] is presented and is often seen in Neural Network examples. There can not be a house priced at $0, hence the implementation of this type of function. A neural network is produced by taking many single neurons and by stacking them together.
Now letś add more variables for the same problem, the representation of the network would be similar to the following graph:



The inter-connecting nods can be seen as individual RELU representations that lead to the values seen on the lines which represent important characteristics in a model and that lead to the final result; X is equal to the stack

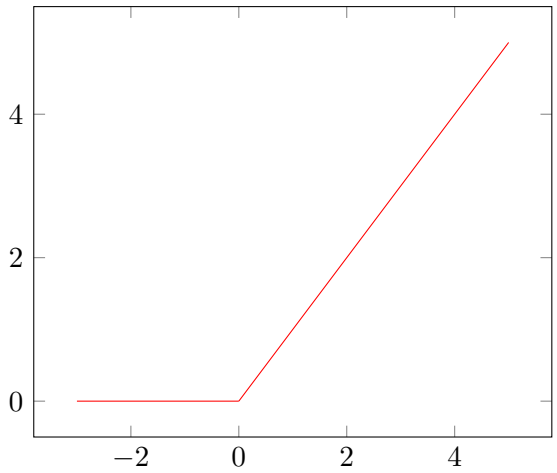


Figure 1.1: Plot of an example RELU function

of the given inputs and Y is equal to the desired output. The "in-between" nods represent hidden features, these are the ones created subsequential to the input data and the network itself defines the relation between these. The network processes two different types of data:

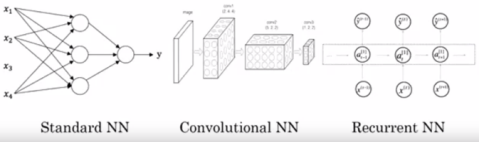
- The X input values
- Training data

With these three the network can predict the output Y value. Giving a Neural Network enough correlational X:Y data will make it better at figuring out the sort of equations that are similar to the ones on the training data.

Supervised Learning

Economic value of deep learning implementations really come from Supervised learning. You have an input x and you want to have a result y. In the previous problem it would be x = Home features, y = Price and the application would impact on the real estate business. Another example could be related to e-marketing, where x = Ad & user info, y = Click on add? (t/f) and the would impact on online advertising. Another examples could be related to computer vision, audio recognition, machine translation and other topic related projects. Simple neural networks such as the two first examples have usually a standard NN implementation, image processing is

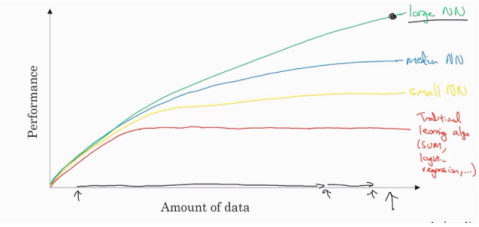
often convolutional and sequential data implementations (such as audio, because it has a temporal component) are often recursive NN; for particular or more complex implementations there can be custom or hybrid architectures.



Structured data (DB with specific fields) vs unstructured data (audio, images, [···]), this tangible difference is important to take into account because historically it has been more difficult to calculate from the latter, hence the importance of the architecture design importance. Although the most "exciting" implementations are often the unstructured data related systems, the market has more demand for well-built structured data solutions.

Why is Deep Learning taking off?

In the traditional learning algorithms, there was a point where the increment of input data did nothing for the performance of the system. In modern problem solving we have a lot of data to handle, so the design of a system which could profit from it was considered crucial. Neural networks are capable of incrementing the system performance in a better fashion than traditional implementations.



One thing to note is that the "Amount of Data" refers only to the amount of labeled data (contains x and y definitions). Also, the notation for the cardinality is expressed as (m).
Although the increment in "learning room" is evident, the performance also has a cap that can occur if you run out of data or if the NN is big enough that it becomes really slow to train.
Large neural nets are only necessary when a lot of data is required to process. There can be little to no difference in large NN and small NN systems where the data-set (m) is relatively small.

Deep learning progress has developed a weighted importance in data, computation-related (hardware-wise) and algorithmic advancements that allow the systems to come to a reality. One of the most notable innovations is the transition from sigmoid to RELU functions; sigmoid functions had areas (where b = 0) that led to the decrease of learning speed due to the slow parameter revision. The innovations of the three disciplines have helped the development cycle (idea -¿ code -¿ experiment) that leads to more architecture revisions.

Week 2

Binary Classification

Usually the data processing is based on going through the entire dataset without explicitly using a for loop. Usually the process consists of a forward pause/forward propagation step, followed by a backward pause/backward propagation step.
Binary classification consists of an input x that is processed by an algorithm that outputs a binary value called label which indicates a certain input behavior or feature. For image classification (cat vs no cat), the image is decomposed to every pixel value per channel and then concatenated into a single object.
A single training example is denoted by the following notation:
$(x, y), x \in \mathbb{R}^{n_x}, y \in \{0, 1\}$; where n_x is the dimension of the input features x.
The training set is comprised of all the pairs of training examples, lowercase m is commonly used as the indicator for the number of training examples.
In order to put the training examples in a more compact fashion, the following model must be used:

$$X = \begin{Bmatrix} \dots & \dots & \dots & \dots \\ x^1 & x^2 & \dots & x^m \\ \dots & \dots & \dots & \dots \end{Bmatrix} \tag{1.1}$$

This matrix X will have m columns, where m is the number of training samples and n_x rows ($X \in \mathbb{R}^{n_x \times m}$). Although there are other implementations, using this convention will make further implementation easier.
For the label output, stacking the results in columns has also been commonly used, and is defined by the following model:

$$Y = \begin{Bmatrix} y^1 & y^2 & \dots & y^m \end{Bmatrix}, Y \in \mathbb{R}^{1 \times m} \tag{1.2}$$

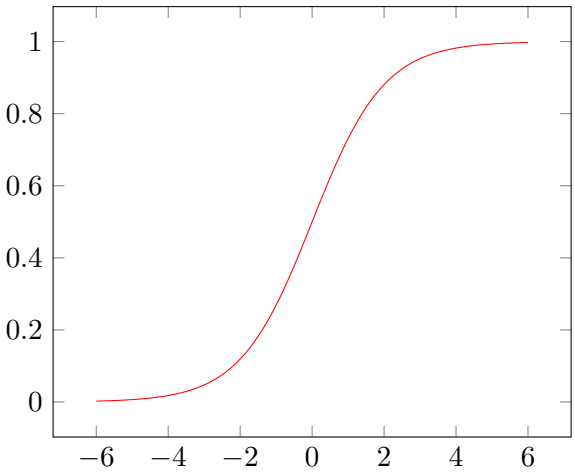


Figure 1.2: Plot of an example sigmoid function

Logistic Regression

Learning algorithm used when output labels Y are either 0 or 1. Given an input feature vector x, you need an algorithm that can output a value $\hat{y}$ representing the prediction weather the feature vector has or has not a certain behavior; the prediction of $\hat{y}$ being equal to 1 can be written as $\hat{y} = P(y = 1|x)$.
Given the parameters $w \in \mathbb{R}^{n_x}, b \in \mathbb{R}$, we must be able to construct a function that goes through the feature vector x and outputs $\hat{y}$. The function is a modification of $w^T x + b$ so that the range does not exceed 1 or be smaller than 0, for this, the use of a sigmoid function is required. The sigmoid of z [1.2] can be calculated through the expression $\sigma(z) = \frac{1}{1+e^{-z}}$; if the z value is very large the output will be closer to 1, on the contrary it will be closer to 0. Given the previous knowledge, we can now assume the following function: $z = w^T x + b, \hat{y} = \sigma(z)$. When implementing logistic regression, it is crucial to learn parameters w and b so that $\hat{y}$ becomes a good estimate of the chance of y being equal to 1; b usually corresponds to an inter-spectrum parameter. In neural network implementations it is recommended to keep these parameters separate although there are some implementations in which these are contained in the same vector.

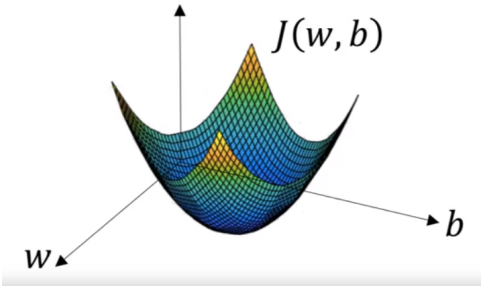


Figure 1.3: Plot of a gradient descent

Logistic Regression Cost Function

To train parameters w and b it is essential to define a cost function. As a result, it is desired that $\hat{y}^{(i)} \approx y^{(i)}$, each training example can be represented as $\hat{y} = \sigma(w^t x^{(i)} + b)$.
To measure the efectiveness of the developed algorithm it is necessary to use the loss function. Square error is often used in other disciplines, but due to the nature of the network-related problems the following formula is used instead: $\mathcal{L}(\hat{y}, y) = -(ylog\hat{y} + (1-y)log(1-\hat{y}))$. The desired result for $\mathcal{L}$ is for it to be as small as possible.
The loss function checks for every training example how it is behaving, in order to check the complete training set a cost function must be used. The cost function is defined by: $J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)})$.

Gradient Descent

The gradient descent algorithm is used to learn the parameters w and b. In the figure [1.3] the parameters are represented in the x and y axis while the result of the cost function J is represented in the z axis. In the figure it's clear that the objective should be to find the values in which w and b converge while the cost function is 0. Analyzing the convex representation of the gradient descent also reveals the use of the previously-mentioned cost function; if square error was utilized, the graph would show crevices leading to erroneous calculations.
The process to estimate the final values consists of an "initial guess" for the parameters, some people use random generators for this step but is quite rare. Gradient descent starts at the given point and "descends" downhill through the graph until it reaches the global optimal.

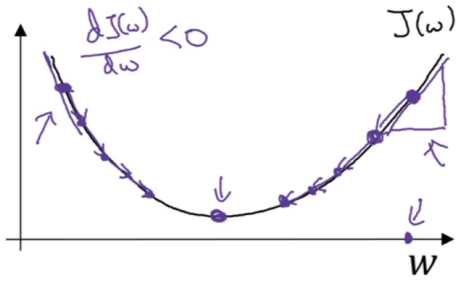


Figure 1.4: Example of a bidimensional gradient descent

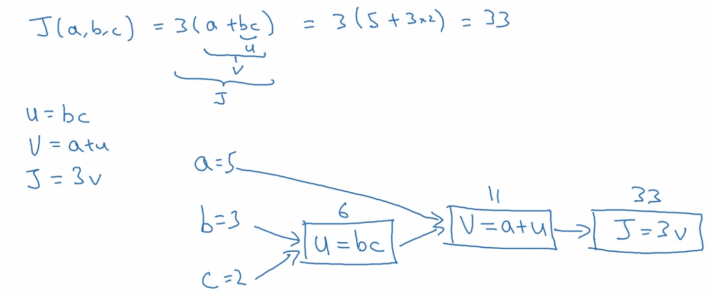
To better understand the gradient descent process let $J(w)$ be a concave function that we want to find the w value for it to be 0; a pseudocode for the descent would be:

```
Repeat{
    w := w - α dJ(w)/dw
}
```

This process is repeated until the algoritm can encounter an acceptable answer for the conversion. α is the representation for the learning rate, this controlls the size of the step for each iteration, and the quantity obtained by the derivative is simply the update of the change that the parameter w will suffer. A visual representation can be the one on figure [1.4].
Of course, thanks to the cost function consisting of two variables the updates would have to occur for both parameters. The updates would be the following for $J(w, b)$:
$w := w - \alpha \frac{\partial J(w,b)}{\partial w}, b := b - \alpha \frac{\partial J(w,b)}{\partial b}$

Computation Graph

Computation of a NNET are organized by forward pass/forward propagation in which the output is calculated, followed by a backward pass/propagation step in which the gradients are computed; the computation graph is a representation in which it's shown why it is done this way. A computation graph is a graphical representation of the calculation processes in a computer. an easy example can be the following:



These are useful when there is an output variable that can be optimized; in logistic regression, the output J of the cost function is expected to be minimized as much as possible. The process for this to be possible is a "right to left" (from the result to the original inputs) that can be translated to derivatives in computing terms.

Derivatives with a Computation Graph

Chapter 2

Improving Deep Neural Networks

Chapter 3

Structure of Machine Learning Project

Chapter 4

Convolutional Neural Networks

Chapter 5

Natural Language Processing