Intro to Deep Learning

Big Data and Machine Learning en el Mercado Inmobiliario
Educación Continua

Ignacio Sarmiento-Barbieri

Universidad de los Andes

November 3, 2022

Deep Learning: Intro

- Neural networks are simple models.
- Their strength lays in their simplicity
- Neural networks combine inputs that are passed through nonlinear activation functions called nodes (or, in reference to the human brain, neurons), to approximate $f^*(x)$

Deep Learning: Intro

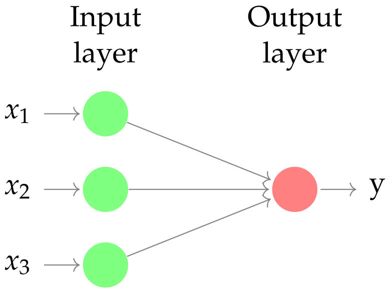
- Let's start with a familiar and simple model, the linear model

$$
\begin{aligned}
y &= f(X) + u \\
y &= \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + u
\end{aligned} \tag{1}
$$

Deep Learning: Intro

- Let's start with a familiar and simple model, the linear model

$$y = f(X) + u \tag{1}$$
$$y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + u$$



Single Layer Neural Networks

- Linear Models may be too simple, and miss the nonlinearities that best approximate $f^*(x)$
- We can overcome these limitations of linear models and handle a more general class of functions by incorporating one or more hidden layers.
- Neural Networks are also called deep feedforward networks, feedforward neural networks, or multilayer perceptrons (MLPs), and are the quintessential deep learning models

Single Layer Neural Networks

- A neural network takes an input vector of p variables

$$X = (X_1, X_2, ..., X_p) \tag{2}$$

- and builds a nonlinear function $f(X)$ to predict the response y .

$$y = f(X) + u \tag{3}$$

- What distinguishes neural networks from previous methods is the particular structure of the model.

Single Layer Neural Networks

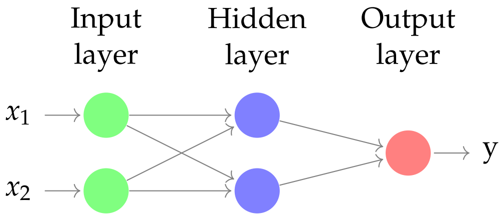
- A NN model has the form

$$
\begin{aligned}
f(X) &= \beta_0 + \sum_{k=1}^{K} \beta_k A_k && (4)\\
&= \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X) && (5)\\
&= \beta_0 + \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right) && (6)
\end{aligned}
$$

- where $g(.)$ is a activiation function specified in advance
- where the nonlinearity of $g(.)$ is essential

Single Layer Neural Networks

- Let's consider a very simple example with
  - $p = 2$, $X = (X_1, X_2)$
  - $K = 2$, $h_1(X)$ and $h_2(X)$
  - $g(z) = z^2$



Single Layer Neural Networks

- Let's consider a very simple example with
  - $p = 2$, $X = (X_1, X_2)$
  - $K = 2$, $h_1(X)$ and $h_2(X)$
  - $g(z) = z^2$
- Then

$$f(X) = \beta_0 + \sum_{k=1}^{2} \beta_k A_k \tag{7}$$

$$= \beta_0 + \sum_{k=1}^{2} \beta_k h_k(X) \tag{8}$$

$$= \beta_0 + \sum_{k=1}^{2} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right) \tag{9}$$

Single Layer Neural Networks

$$f(X) = \beta_0 + \sum_{k=1}^{2} \beta_k g\left(w_{k0} + \sum_{j=1}^{2} w_{kj} X_j\right) \tag{10}$$

- We specify the parameters as

$$\begin{array}{lll} \beta_0 = 0 & \beta_1 = \frac{1}{4} & \beta_2 = -\frac{1}{4} \\ w_{10} = 0 & w_{11} = 1 & w_{12} = 1 \\ w_{20} = 0 & w_{21} = 1 & w_{22} = -1 \end{array}$$

Single Layer Neural Networks

- Then

$$h_1(X) = (0 + X_1 + X_2)^2 \tag{11}$$
$$h_2(X) = (0 + X_1 - X_2)^2 \tag{12}$$

- and plugging in

$$f(X) = 0 + \frac{1}{4}(0 + X_1 + X_2)^2 - \frac{1}{4}(0 + X_1 - X_2)^2 \tag{13}$$
$$= \frac{1}{4}\left((X_1 + X_2)^2 - (X_1 - X_2)^2\right) \tag{14}$$
$$= X_1 X_2 \tag{15}$$

Worked Example: The ”Exclusive OR (XOR)” Function

- The exclusive disjunction of a pair of propositions, (p, q), is supposed to mean that p is true or q is true, but not both
- It’s truth table is:

q	p	q v p
0	0	0
0	1	1
1	0	1
1	1	0

- When exactly one of these binary values is equal to 1, the XOR function returns 1. Otherwise, it returns 0

Worked Example: The "Exclusive OR (XOR)" Function

- Let's use a linear model

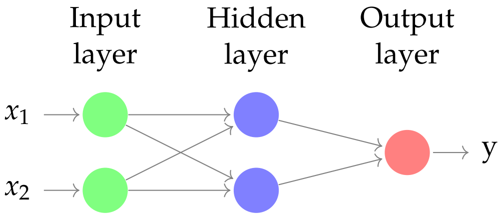
$$y = X\beta + \iota\alpha \tag{16}$$

$$y = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} X = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \iota = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \tag{17}$$

- Solution $\alpha = \frac{1}{2}$ $\beta = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$
- Prediction $\hat{y} = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$

Worked Example: The "Exclusive OR (XOR)" Function

- Let's use Single Layer NN containing two hidden units



Worked Example: The "Exclusive OR (XOR)" Function

- Which activation functions (A_k) should we choose?
  - Clearly **not** linear, otherwise it would defeat the entire purpose
  - We are going to use the rectified linear unit or ReLU (it is usually the default recommendation, there are many others (more on this later))
  - ReLU is defined as $g(z) = max\{0, z\}$
- For the output layer? For this example, a linear model will suffice

$$f(X) = \beta_0 + w_k A_k \tag{18}$$

- The final model is then

$$f(x, W, C, w, b) = max\{0, XW + c\}\, w + b \tag{19}$$

Worked Example: The "Exclusive OR (XOR)" Function

- Suppose this is the solution to the XOR problem

$$f(x) = max\{0, XW + c\}\, w + b$$

$$W = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$c = \begin{pmatrix} 0 & -1 \\ 0 & -1 \\ 0 & -1 \\ 0 & -1 \end{pmatrix}$$

$$w = \begin{pmatrix} 1 & -2 \end{pmatrix}$$

$$b = 0$$

Worked Example: The "Exclusive OR (XOR)" Function

- Lets work out the example step by step

$$f(x) = max\{0, XW + c\}\, w + b \tag{20}$$

$$XW = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{pmatrix}$$

$$XW + c = \begin{pmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

$$max\{0, XW + c\} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

Worked Example: The "Exclusive OR (XOR)" Function

$$\hat{y} = max\{0, XW + c\}\, w + b = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

- The neural network has obtained the correct answer for every data point

Worked Example: The "Exclusive OR (XOR)" Function

- In this example, we simply specified the solution, then showed that it obtained zero error.
- In a real situation, obviously we can't guess the solution
- What we do is gradient based optimization
- Remember that the convergence point of gradient descent depends on the initial values of the parameters and step size.
- In practice, gradient descent would usually not find clean, easily understood, integer-valued solutions like we did here.

NN Minimalist Theory

- Why not a linear activation functions?
- Let's go back to our example
  - $p = 2$, $X = (X_1, X_2)$
  - $K = 2$, $h_1(X)$ and $h_2(X)$
  - Now $g(z) = z$
- Then

$$\begin{aligned} f(X) &= \beta_0 + \sum_{k=1}^{2} \beta_k A_k & (21) \\ &= \beta_0 + \sum_{k=1}^{2} \beta_k h_k(X) & (22) \\ &= \beta_0 + \sum_{k=1}^{2} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right) & (23) \end{aligned}$$

NN Minimalist Theory

Why not a linear activation functions?

- Since $g(z) = z$ we get

$$f(X) = \beta_0 + \sum_{k=1}^{2} \beta_k \left(w_{k0} + \sum_{j=1}^{2} w_{kj} X_j \right) \tag{24}$$

- Replacing

$$f(X) = \beta_0 + \beta_1 \left(w_{10} + w_{11} X_1 + w_{12} X_2 \right) + \beta_2 \left(w_{20} + w_{21} X_1 + w_{22} X_2 \right) \tag{25}$$

- then

$$f(X) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 \tag{26}$$

Activation Functions

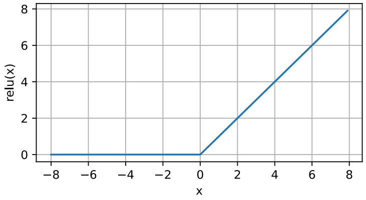
- The gain comes from using nonlinear activation function f
- Note that, with nonlinear activation functions in place, it is no longer possible to collapse our NN into a linear model.
- Activation functions are fundamental to deep learning, let us briefly survey some common activation functions.
- In practice we would not use a quadratic function, since we would always get a second-degree plynomial in the original coordinates
- We use others that we brefly review here

Activation Functions

ReLU Function

- ReLU Function
  - The most popular choice, due to both simplicity of implementation and its good performance on a variety of predictive tasks, is the rectified linear unit (ReLU).
  - ReLU provides a very simple nonlinear transformation. Given an element x, the function is defined as the maximum of that element and 0:

$$\text{ReLU}(x) = \max\{x, 0\}.$$

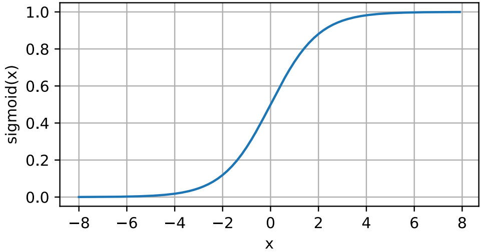


Activation Functions

Sigmoid Function (Logit)

- The sigmoid function transforms its inputs, for which values lie in the domain $\mathbb{R}$, to outputs that lie on the interval (0, 1).
- For that reason, the sigmoid is often called a squashing function: it squashes any input in the range (-inf, inf) to some value in the range (0, 1):

$$\operatorname{sigmoid}(x) = \frac{1}{1 + \exp(-x)}.$$



- In the earliest neural networks, scientists were interested in modeling biological neurons which either fire or do not fire. Thus the pioneers of this field,

Activation Functions

- Other Activation functions
  - Tanh: $\tanh(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}$.
  - Hard tanh: $max(-1, min(1, x))$
  - Radial basis function (RBF): $exp\left(\frac{1}{\sigma^2)}||W - x||^2\right)$
  - Softplus: $log(1 + e^x)$
  - $h = cos(Wx + b)$ Goodfellow et al. (2016) claim that on the MNIST dataset they obtained an error rate of less than 1 percent
- Hidden unit design remains an active area of research, and many useful hidden unit types remain to be discovered

Output Functions

- The choice of cost function is tightly coupled with the choice of output unit.
- Most of the time, we simply use the distance between the data distribution and the model distribution.
  - Linear $y = W'h + b \rightarrow \mathbb{R}$
  - Sigmoid (Logistic) $\frac{1}{1+\exp(-x)} \rightarrow$ classification $\{0, 1\}$
  - Softmax $\frac{exp(x)}{\sum exp(x))} \rightarrow$ classification multiple categories

Architecture Design

- Another key design consideration for neural networks is determining the architecture.
- The word architecture refers to the overall structure of the network: how many units it should have and how these units should be connected to each other.
- The universal approximation theorem guarantees that regardless of what function we are trying to learn, a sufficiently large MLP will be able to represent this function.

Architecture Design

- We are not guaranteed, however, that the training algorithm will be able to learn that function.
- Even if the network is able to represent the function, learning can fail for two different reasons.
  1. The optimization algorithm used for training may not be able to find the value of the parameters that corresponds to the desired function.
  2. The training algorithm might choose the wrong function as a result of overfitting

Architecture Design

- A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasible large and may fail to learn and generalize correctly.
- In many circumstances, using deeper models can reduce the number of units required to represent the desired function and can reduce the amount of generalization error.
- The ideal network architecture for a task must be found via experimentation guided by monitoring the validation set error

Further Readings

- Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola (2020) Dive into Deep Learning. Release 0.15.1. http://d2l.ai/index.html
- Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1, No. 2). Cambridge: MIT press. http://www.deeplearningbook.org
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). An introduction to statistical learning.
- Rstudio (2020). Tutorial TensorFlow https://tensorflow.rstudio.com/tutorials/beginners/basic-ml/tutorial_basic_classification/
- Taddy, M. (2019). Business data science: Combining machine learning and economics to optimize, automate, and accelerate business decisions. McGraw Hill Professional.