Intro to Deep Learning

Big Data y Machine Learning para Economía Aplicada

Ignacio Sarmiento-Barbieri

Universidad de los Andes

Agenda

Deep Learning: Intro

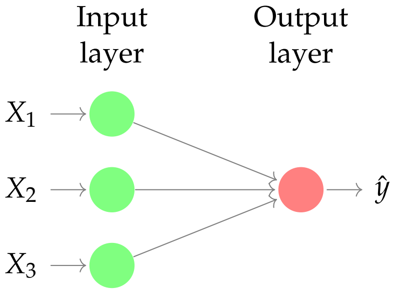
- Linear Models may miss the nonlinearities that best approximate $f^*(x)$
- Neural networks are simple models.
- The model has **linear combinations** of inputs that are passed through **nonlinear activation functions** called nodes (or, in reference to the human brain, neurons).

Deep Learning: Intro

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

Deep Learning: Intro

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



Single Layer Neural Networks

- A neural network takes an input vector of p variables

$$X = (X_1, X_2, ..., X_p) \tag{1}$$

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

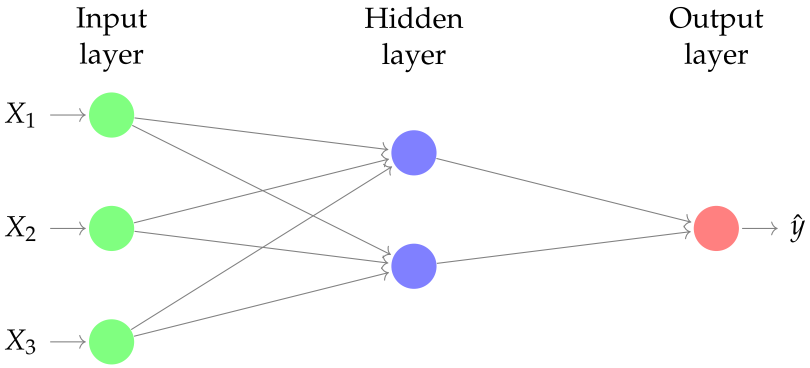
$$y = f(X) + u \tag{2}$$

- The Single layer NN model has the form

$$f(X) = f^{(output)}(g(X)) \tag{3}$$

- where g is the activation function in the hidden unit
- the second layer, $f^{(output)}$ is the output layer of the network

Single Layer Neural Networks



Single Layer Neural Networks

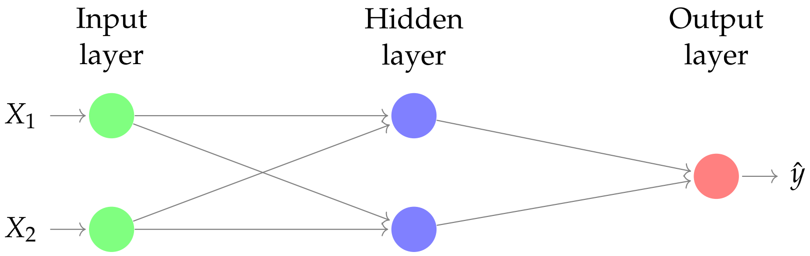
- NN are made of **linear combinations** of inputs that are passed through **nonlinear activation functions**

Worked Example I: Single Layer Neural Networks

- 2 Predictors: $p = 2$, $X = (X_1, X_2)$
- 2 Nodes: $K = 2$, $A_1(X)$ and $A_2(X)$
- Non-linear activation function $g(z) = z^2$
- Want to predict a number $\in \mathbb{R}$: identity output function ($f^{(output)} : \mathbb{R} \rightarrow \mathbb{R}$ such that $f^{(output)}(x) = x$)

Worked Example I: Single Layer Neural Networks

- 2 Predictors: $p = 2$, $X = (X_1, X_2)$
- 2 Nodes: $K = 2$, $A_1(X)$ and $A_2(X)$
- Non-linear activation function $g(z) = z^2$
- Want to predict a number $\in \mathbb{R}$: identity output function ($f^{(output)} : \mathbb{R} \to \mathbb{R}$ such that $f^{(output)}(x) = x$)



Why not linear activation functions?

Worked Example II : 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	y
0	0	0
0	1	1
1	0	1
1	1	0

Agenda

Neural Networks: Activation Functions

- $\text{Sigmoid}(x) = \frac{1}{1+\exp(-x)}$
- $\text{ReLU}(x) = \max\{x, 0\}$
- Among others (see more here)
- Hidden unit design remains an active area of research, and many useful hidden unit types remain to be discovered

Activation Functions

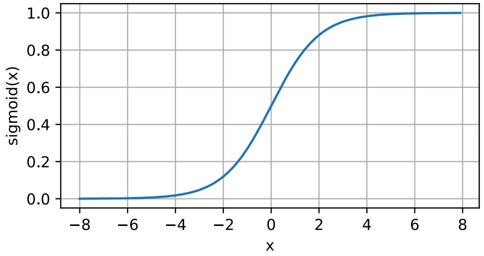
Sigmoid Function (Logit)

- 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):

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

Activation Functions

Sigmoid Function (Logit)



- When attention shifted to gradient based learning, the sigmoid function was a natural choice because it is smooth and differentiable.

Activation Functions

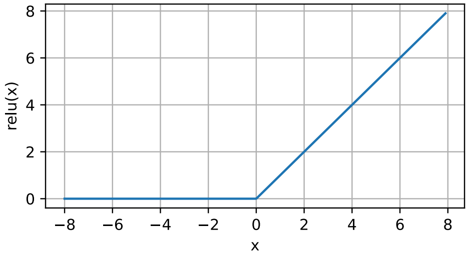
ReLU Function

- 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

- ReLU function retains only positive elements and discards all negative elements by setting them to 0.
- It is piecewise linear.



Agenda

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 $\rightarrow \mathbb{R}$egression
  - Sigmoid (Logistic) $\rightarrow$ classification $\{0, 1\}$
  - Softmax $\rightarrow$ classification multiple categories

Loss Functions

- We want to estimate

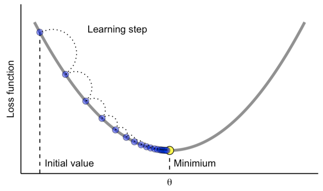
$$y_i = f(x) + u$$

- The first thing we need to specify is the loss function.
- For regression problems we used MSE.

$$E(\theta) = \frac{1}{N}\sum(y - \hat{y})^2 \tag{4}$$

Gradient-Based Descent

- The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.
- Intuitively, the method corresponds to “walking down the hill” in our many parameter landscape until we reach a (local) minimum.



Source: Boehmke, B., & Greenwell, B. (2019)

Batch Gradient Descent

- To implement Gradient Descent, you need to compute the gradient of the cost function with regards to each model parameter

$$\beta' = \beta - \epsilon \nabla_{\beta} f(\beta) \tag{5}$$

- In other words, you need to calculate how much the cost function will change if you change β just a little bit.
- You also need to define the learning step ϵ

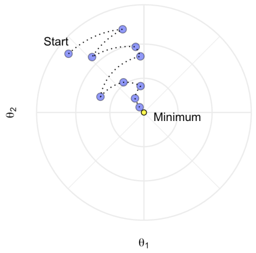
Batch Gradient Descent

- Notice that this formula involves calculations over the full data set X, at each Gradient Descent step!
- This is why the algorithm is called Batch Gradient Descent: it uses the whole batch of training data at every step.
- As a result it is terribly slow on very large data sets.
- However, Gradient Descent scales well with the number of variables; estimating a Linear Regression model when there are hundreds of thousands of features is much faster using Gradient Descent than using the Normal Equations or any decomposition.

Stochastic Gradient-Based Optimization

- Using the full data set can be terribly slow, specially in large data sets.
- At the opposite extreme, Stochastic Gradient Descent just picks a random observation at every step and computes the gradients based only on that single observation.

Stochastic Gradient-Based Optimization



Source: Boehmke, B., & Greenwell, B. (2019)

- This makes the algorithm faster but
  - Adds some random nature in descending the risk function's gradient.
  - Although this randomness does not allow the algorithm to find the absolute global minimum, it can actually help the algorithm jump out of local minima and off plateaus to get sufficiently near the global minimum.

Mini-batch Gradient Descent

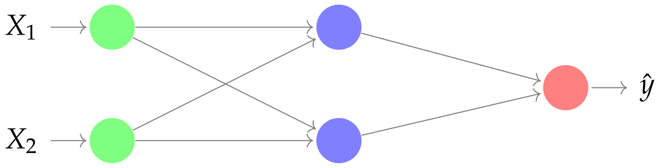
- At each step, instead of computing the gradients based on the full dataset (as in Batch GD) or based on just one observation (as in Stochastic GD),
- Mini- batch GD computes the gradients on small random sets of observations called mini- batches.

Agenda

Training the network

Example: House Prices

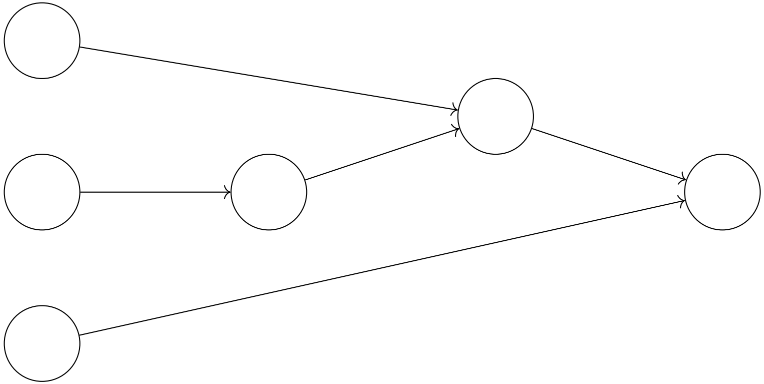
Training the network



Backpropagation and Computational Graphs

- We need the relevant gradients for each weight, the derivative of the loss with respect to each weight in every layer of the network.
- **Problem** The loss is computed only at the very end of the network.
- How do we find these gradients for weights in the early layers?
- The solution is a method called error back propagation
- It is a special case of backward differentiation, a method that relies on computation graphs.
- A computation graph is a representation of the process of computing any mathematical expression in which we break down the computation into separate operations, each of which is modeled as a node in a graph.

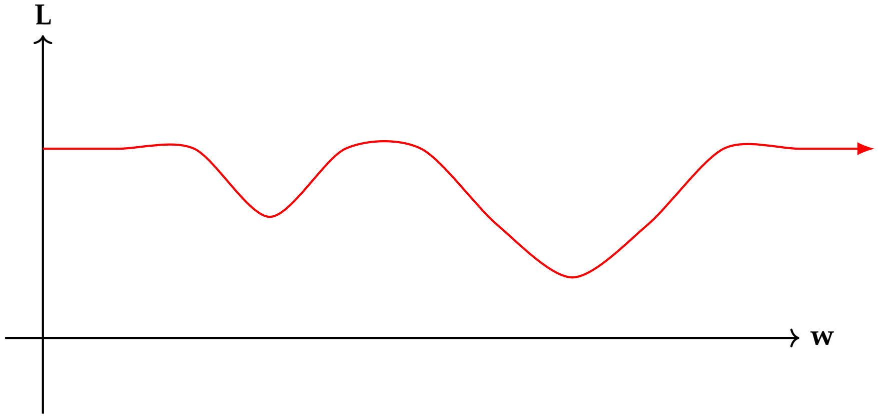
Backpropagation and Computational Graphs



Optimization

- SGD (Robbins and Monro, 1951)
- SGD + momentum (Polyak, 1964)
- SGD + nesterov momentum (Nesterov, 1983; Sutskever et al. 2013)
- Ada Grad (Duchi, 2011)
- RMSProp (Tieleman and Hinton, 2012)
- Adam (Kingma and Ba, 2014)

Optimization



Training the network

Example: MNIST

Agenda

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 (Hornik et al., 1989; Cybenko, 1989) guarantees that regardless of what function we are trying to learn, a sufficiently large MLP will be able to represent this function.

Architecture Design

- The universal approximation theorem (Hornik et al., 1989; Cybenko, 1989) states that:
  - A feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation function (such as the logistic sigmoid activation function) can approximate any Borel measurable function from one finite-dimensional space to another with any desired nonzero amount of error, provided that the network is given enough hidden units.

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

Agenda

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

Multilayer Neural Networks

- Modern neural networks typically have more than one hidden layer, and often many units per layer.
- In theory a single hidden layer with a large number of units has the ability to approximate most functions.
- However, the learning task of discovering a good solution is made much easier with multiple layers each of modest size.

Example: MNIST

Agenda

Network Tuning

- Training networks requires a number of choices that all have an effect on the performance:
  - The number of hidden layers,
  - The number of units per layer
  - Regularization tuning parameters
  - Details of stochastic gradient descent.
- This is an active research area that involves a lot of trial and error, and overfitting is a latent danger at each step.

Agenda

When to Use Deep Learning?

- The performance of deep learning ussually is very impressive.
- The question that then begs an answer is: should we discard all our older tools, and use deep learning on every problem with data?
- Let's see an example

When to Use Deep Learning?