Intro to Deep Learning

Big Data y Machine Learning para Economía Aplicada

Ignacio Sarmiento-Barbieri

Universidad de los Andes

Agenda

1. Single Layer Neural Networks
2. Activation Functions
3. Output Functions
4. Training the network
5. Architecture Design
6. Multilayer Neural Networks
7. Network Tuning
8. When to Use Deep Learning?

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).

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 .

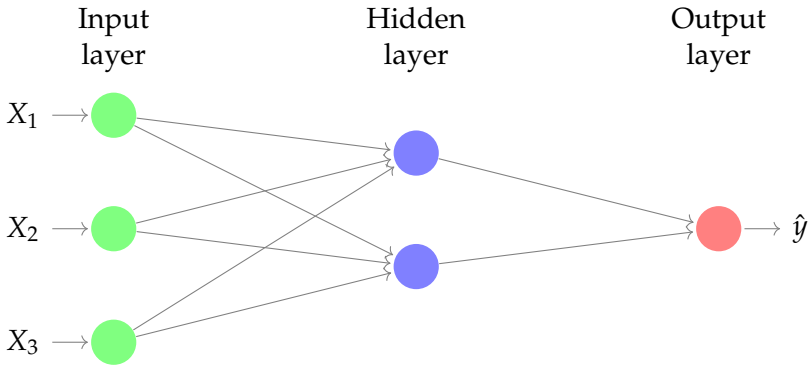
$$y = f(X) + u \tag{2}$$

- 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**
- The NN model has the form

$$f(X) = f^{(output)}\left[\beta_0 + \sum_{k=1}^{K} \beta_k A_k\right] \tag{4}$$

$$= f^{(output)}\left[\beta_0 + \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right)\right] \tag{5}$$

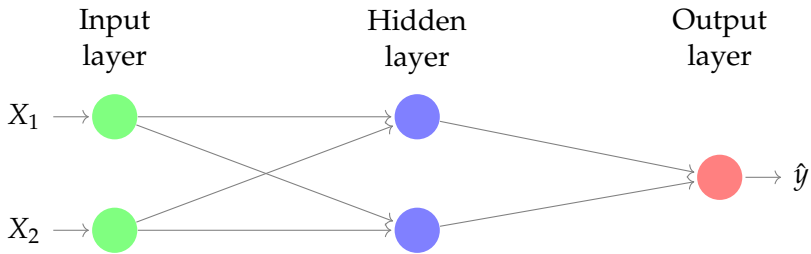
- where
  - $g(.)$ is a activiation function, the nonlinearity of $g(.)$ is **key**
  - $f^{(output)}$ is the output layer of the network
- both are prespecified

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 $f^{(output)}(x) = x$)

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 $f^{(output)}(x) = x$)



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 $f^{(output)}(x) = x$)

$$f(X) = f^{(output)}(g(X)) \tag{6}$$

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 $f^{(output)}(x) = x$)

$$f(X) = f^{(output)}(g(X)) \tag{6}$$

$$f(X) = \beta_0 + \sum_{k=1}^{2} \beta_k \left(w_{k0} + \sum_{j=1}^{2} w_{kj} X_j \right)^2 \tag{7}$$

Worked Example I: Single Layer Neural Networks

- Suppose we get

$$\begin{array}{lll} \hat{\beta}_0 = 0 & \hat{\beta}_1 = \frac{1}{4} & \hat{\beta}_2 = -\frac{1}{4} \\ \hat{w}_{10} = 0 & \hat{w}_{11} = 1 & \hat{w}_{12} = 1 \\ \hat{w}_{20} = 0 & \hat{w}_{21} = 1 & \hat{w}_{22} = -1 \end{array}$$

- Then

$$A_1(X) = (0 + X_1 + X_2)^2 \tag{8}$$

$$A_2(X) = (0 + X_1 - X_2)^2 \tag{9}$$

- and plugging in

$$f(X) = 0 + \frac{1}{4}(0 + X_1 + X_2)^2 - \frac{1}{4}(0 + X_1 - X_2)^2 \tag{10}$$

$$= \frac{1}{4}\left((X_1 + X_2)^2 - (X_1 - X_2)^2\right) \tag{11}$$

$$= X_1 X_2 \tag{12}$$

Why not linear activation functions?

- Supose now that $g(z) = z$

$$f(X) = \beta_0 + \sum_{k=1}^{2} \beta_k \left(w_{k0} + \sum_{j=1}^{2} w_{kj} X_j \right) \tag{13}$$

- Replacing

$$f(X) = \beta_0 + \beta_1 \left(w_{10} + w_{11} X_1 + w_{12} X_2 \right) + \beta_2 \left(w_{20} + w_{21} X_1 + w_{22} X_2 \right) \tag{14}$$

- then

$$f(X) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 \tag{15}$$

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	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

$$y = \beta_0 + \beta_1 q + \beta_2 p + u \tag{16}$$

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

- Solution $\hat{\beta}_0 = \frac{1}{2}, \hat{\beta}_1 = 0, \hat{\beta}_2 = 0$
- Prediction $\hat{y} = \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}$
- $MSE = 1$

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

- Let's use Single Layer NN containing two hidden units
- Activation Funcition: ReLU: $g(z) = max\{0, z\}$
- NN

$$f(X) = \beta_0 + \sum_{k=1}^{2} \beta_k g\left(w_{k0} + \sum_{j=1}^{2} w_{kj} X_j\right) \tag{18}$$

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

- Suppose this is the solution to the XOR problem

$$f(x) = max\{0, XW + W_0\}\,\beta + \beta_0$$

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

$$W_0 = \iota_4 \begin{pmatrix} 0 & -1 \end{pmatrix}$$

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

$$\beta_0 = 0$$

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

- Lets work out the example step by step

$$f(x) = max\{0, XW + W_0\}\,\beta + \beta_0 \tag{19}$$

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

$$XW + W_0 = \begin{pmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

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

$$max\{0, XW + W_0\} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

$$\hat{y} = max\{0, XW + W_0\}\,\beta + \beta_0 = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -2 \end{pmatrix} + 0 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Agenda

1. Single Layer Neural Networks
2. Activation Functions
3. Output Functions
4. Training the network
5. Architecture Design
6. Multilayer Neural Networks
7. Network Tuning
8. When to Use Deep Learning?

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

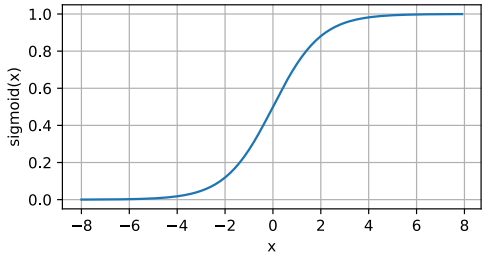
Sigmoid Function (Logit)

- 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

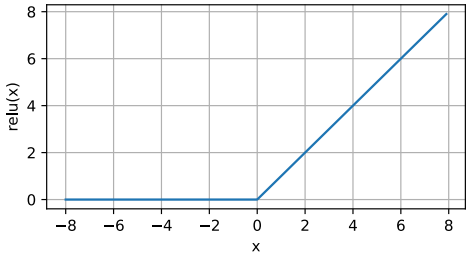
ReLU Function

- 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

1. Single Layer Neural Networks
2. Activation Functions
3. Output Functions
4. Training the network
5. Architecture Design
6. Multilayer Neural Networks
7. Network Tuning
8. When to Use Deep Learning?

Output Functions

- The choice of output unit is related to the problem at hand
  - Regression:

$$f^{(output)} : \mathbb{R} \rightarrow \mathbb{R} \tag{20}$$

  - Classification:

$$f^{(output)} : \mathbb{R} \rightarrow [0, 1] \tag{21}$$

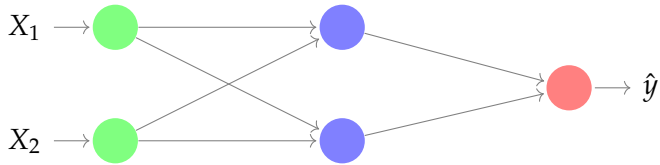
Agenda

1. Single Layer Neural Networks
2. Activation Functions
3. Output Functions
4. Training the network
5. Architecture Design
6. Multilayer Neural Networks
7. Network Tuning
8. When to Use Deep Learning?

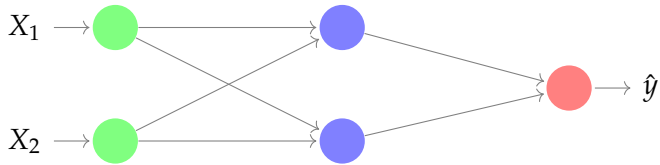
Training the network

Example: House Prices

Training the network



Training the network



- Equations
  - Hidden Layer sigmoid (logistic):
    - $A_1 = \sigma(w_{11} \cdot X_1 + w_{12} \cdot X_2 + w_{10})$
    - $A_2 = \sigma(w_{21} \cdot X_1 + w_{22} \cdot X_2 + w_{20})$
  - Output Layer, identity output function:
    - $y = \beta_0 + \beta_1 \cdot A_1 + \beta_2 \cdot A_2$
- Loss Function $\Rightarrow$ MSE: $\frac{1}{2}(y - \hat{y})^2$

Training the network

Backpropagation

- For simplicity let's focus on updating w_{11}

$$w'_{11} = w_{11} - \epsilon \cdot \frac{\partial \text{Loss}}{\partial w_{11}} \tag{22}$$

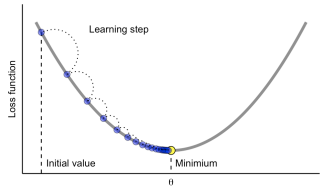
- Chain rule

$$\frac{\partial \text{Loss}}{\partial w_{11}} = \frac{\partial \text{Loss}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial A_1} \frac{\partial A_1}{\partial w_{11}} \tag{23}$$

1. $\frac{\partial \text{Loss}}{\partial y} = -(y - \hat{y})$
2. $\frac{\partial y}{\partial A_1} = \beta_1$
3. $\frac{\partial A_1}{\partial w_{11}} = \sigma'(w_{11} \cdot X_1 + w_{12} \cdot X_2 + b_1) \cdot X_1$

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)

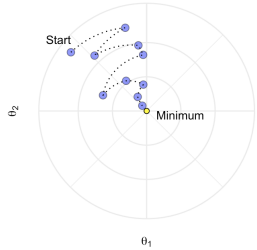
Batch Gradient Descent

- 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.

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
- 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.
- The main advantage of Mini-batch GD over Stochastic GD is that you can get a performance boost from distributed computing and hardware optimization of matrix operations, especially when using GPUs.

Training the network

Example: MNIST

Agenda

1. Single Layer Neural Networks
2. Activation Functions
3. Output Functions
4. Training the network
5. Architecture Design
6. Multilayer Neural Networks
7. Network Tuning
8. When to Use Deep Learning?

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.

Agenda

1. Single Layer Neural Networks
2. Activation Functions
3. Output Functions
4. Training the network
5. Architecture Design
6. Multilayer Neural Networks
7. Network Tuning
8. When to Use Deep Learning?

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

1. Single Layer Neural Networks
2. Activation Functions
3. Output Functions
4. Training the network
5. Architecture Design
6. Multilayer Neural Networks
7. Network Tuning
8. When to Use Deep Learning?

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

1. Single Layer Neural Networks
2. Activation Functions
3. Output Functions
4. Training the network
5. Architecture Design
6. Multilayer Neural Networks
7. Network Tuning
8. When to Use Deep Learning?

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?