Intro to Deep Learning

Big Data y Machine Learning para Economía Aplicada

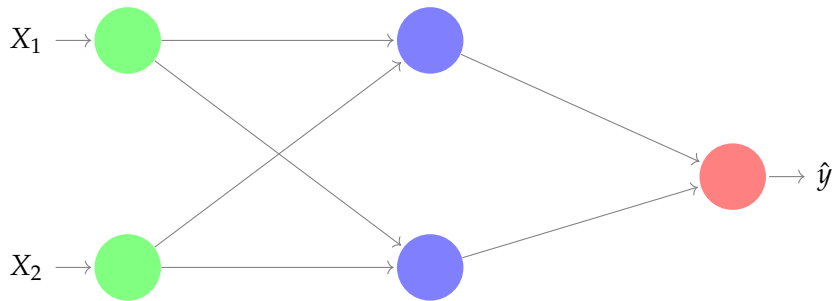Ignacio Sarmiento-Barbieri

Universidad de los Andes

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).

Agenda

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

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

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

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

Agenda

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 ∨ 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

$$y = \beta_0 + \beta_1 q + \beta_2 p + u \tag{3}$$

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

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

- Activation Function: ReLU: $g(z) = max\{0, z\}$

- NN

$$f(X) = \beta_0 + \sum_{k=1}^{2} \beta_k g\left(w_{k0} + \sum_{j=1}^{2} w_{kj}X_j\right) \quad (4)$$

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

Agenda

1. Recap: SNN
   - Example: XOR
   - Activation Functions
   - Output Functions

2. Training the network

3. Architecture Design
   - Deep Neural Networks

4. When to Use Deep Learning?

Neural Networks: Activation Functions

▶ Sigmoid$(x) = \frac{1}{1+\exp(-x)}$

▶ 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

Agenda

Output Functions

► The choice of output unit is related to the problem at hand

  ► Regression

  ► Classification

    ► Binary

    ► Multiclass

Agenda

Training the network

▶ El objetivo es

$$\hat{f} = \underset{f}{\mathrm{argmin}} \left\{ \sum_{i=1}^{n} L(y, f(X; \Theta)) \right\} \tag{5}$$

▶ SNN

$$f(X, \beta, w) = f\left[\beta_0 + \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j \right) \right] \tag{6}$$
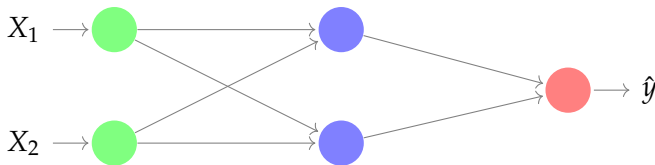
Training the network

Example: House Prices

Training the network

Example: House Prices

- ▶ 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:
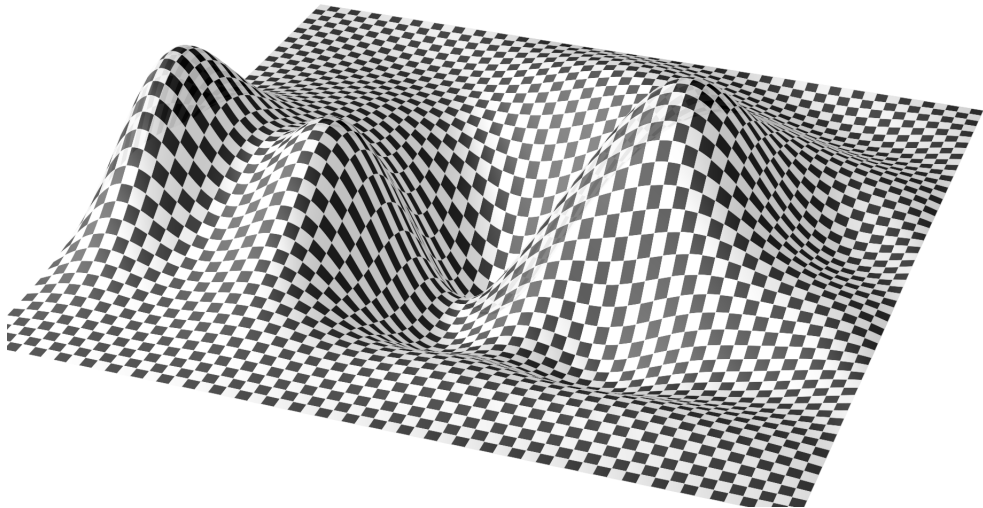   - ▶ $\hat{y}_i = \beta_0 + \beta_1 \cdot A_1 + \beta_2 \cdot A_2$

- ▶ Loss Function $\Rightarrow$ MSE: $\frac{1}{n} \sum_i^n (y_i - \hat{y}_i)^2$
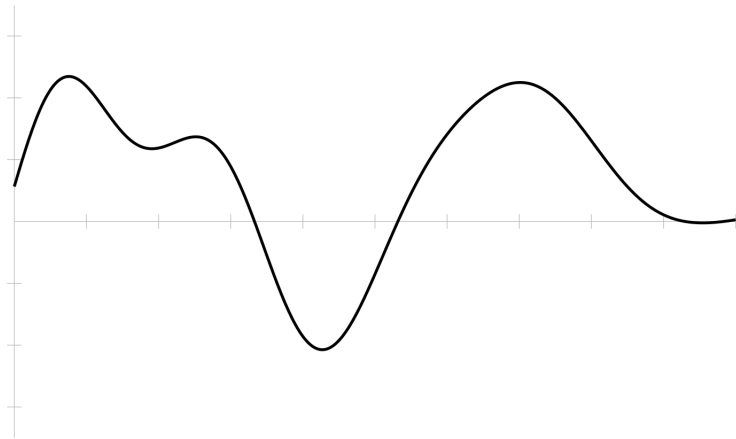
Training the network

► El objetivo es

$$\hat{f} = \underset{w,\beta}{\mathrm{argmin}} \left\{ \sum_{i=1}^{n} L(y, f \left[\beta_0 + \sum_{k=1}^{K} \beta_k g \left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j \right) \right]) \right\} \tag{7}$$
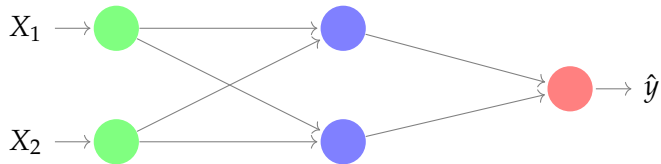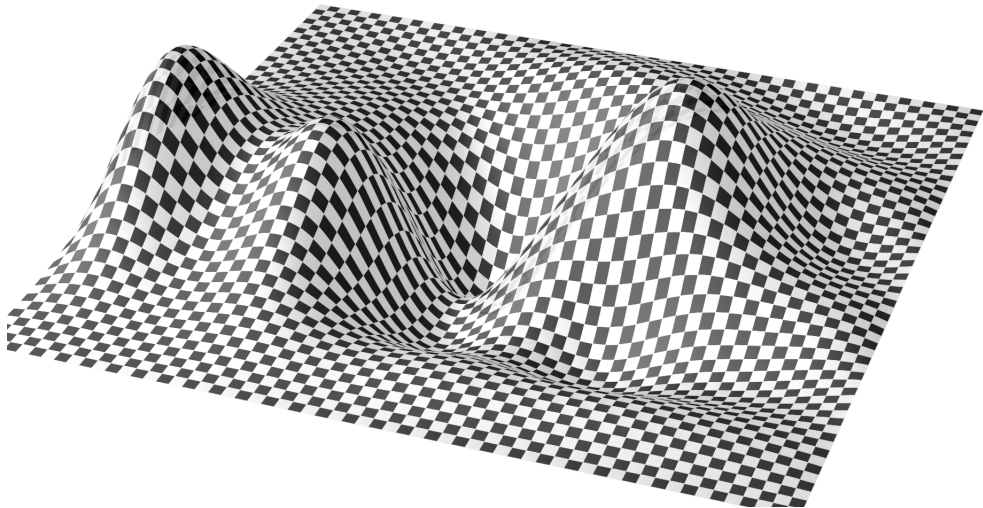
Training the network

Training the network

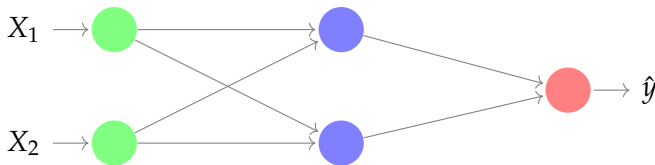# Training the network

Training the network

Training the network
Backpropagation

$$\theta' = \theta - \epsilon \cdot \nabla_\theta \mathcal{L}(\theta)$$

Training the network

Example: House Prices

- ▶ 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:
        - ▶ $\hat{y}_i = \beta_0 + \beta_1 \cdot A_1 + \beta_2 \cdot A_2$

- ▶ Loss Function $\Rightarrow$ MSE: $\frac{1}{n} \sum_i^n (y_i - \hat{y}_i)^2$

How does backpropagation work?

Updating a single weight

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

Training the network
Updating a weight in the output layer

- ▶ Now let's update one of the weights from the hidden layer to the output layer, β_1
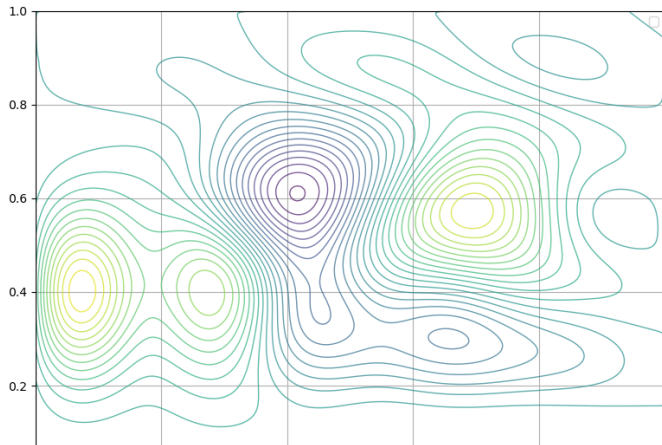
Training the network
Batch Gradient Descent

- ▶ Notice that this formula involves calculations over the full data set, 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.

Training the network
Stochastic Gradient-Based Optimization

- ▶ Stochastic Gradient Descent just picks a random observation at every step and computes the gradients based only on that single observation.
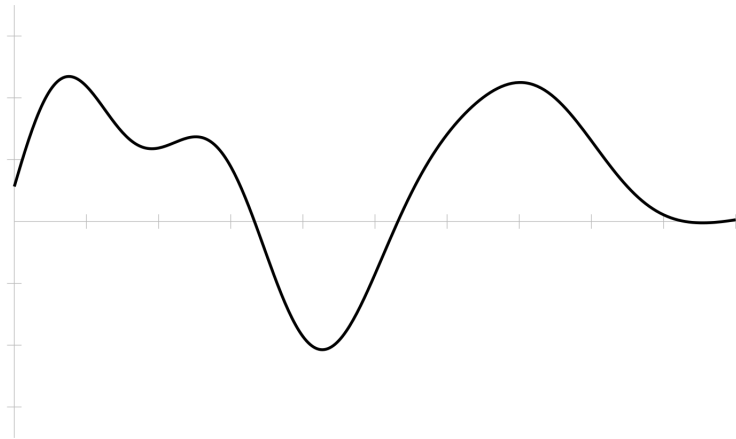
Training the network
Mini-batch Gradient Descent

- ▶ Batch Gradient Descent involves calculations over the full data set

- ▶ Stochastic Gradient Descent just picks one random observation at every step

- ▶ At each step, mini- batch GD computes the gradients on small random sets of observations called mini- batches.
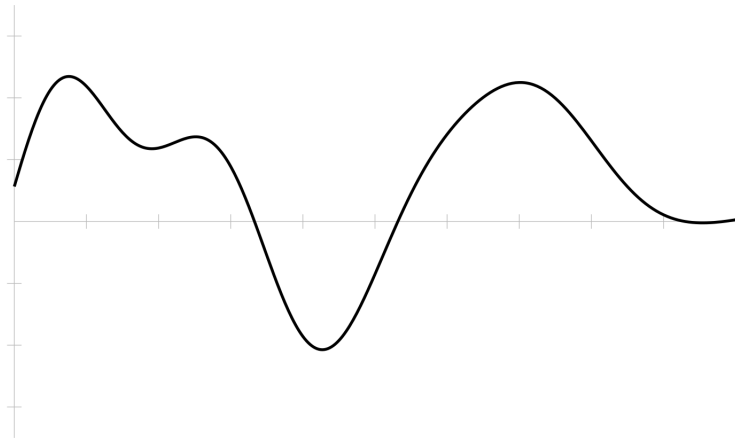
Training the network

Training the network
SGD + Momentum, Polyak, 1964

$$v' = \gamma v - \epsilon \cdot \nabla_\theta \mathcal{L}(\theta)$$
$$\theta' = \theta + v'$$

▶ Agrega una "inercia" al gradiente.
▶ Permite acumular dirección → suaviza la trayectoria.
▶ Ayuda a escapar de mínimos poco profundos.

Training the network

Training the network
RMSProp, Tieleman and Hinton, 2012

$$E[g^2] = \rho E[g^2]_{iter-1} + (1-\rho)g^2$$
$$\theta' = \theta - \frac{\epsilon}{\sqrt{E[g^2] + \epsilon}} \cdot g_t$$

- ▶ Variante de AdaGrad con memoria exponencial.
- ▶ Controla mejor la escala del gradiente.
- ▶ Muy usado en redes neuronales profundas.

Training the network
Adam, Kingma and Ba, 2014

$$m = \beta_1 m_{iter-1} + (1 - \beta_1)g$$

$$v = \beta_2 v_{iter-1} + (1 - \beta_2)g^2$$

$$\hat{m} = \frac{m}{1 - \beta_1^{iter}}, \quad \hat{v} = \frac{v}{1 - \beta_2^{iter}}$$

$$\theta' = \theta - \epsilon \cdot \frac{\hat{m}}{\sqrt{\hat{v}} + \eta}$$

- ▶ Combina Momentum + RMSProp.
- ▶ Corrige sesgo en los primeros pasos.
- ▶ Muy popular por su robustez y rapidez.

Training the network
Which one to use? Zhou et al., 2020

- ▶ Aunque Adam converge más rápido, SGD suele encontrar soluciones que generalizan mejor.

- ▶ Zhou et al., 2020 modela ambos métodos

- ▶ SGD: Puede

    - ▶ Escapar más fácilmente de mínimos locales.
    - ▶ Alcanzar soluciones que generalizan mejor.

- ▶ Adam: quedar atrapado en mínimos subóptimos.

Conclusión: El ruido inherente al SGD actúa como una forma de regularización que favorece la generalización.

Training the network
Example: MNIST

Agenda

Architecture Design

▶ A 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

- ▶ A 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.

- ▶ However, 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

- 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

Agenda

Architecture Design
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.

Architecture Design

Multilayer Neural Networks
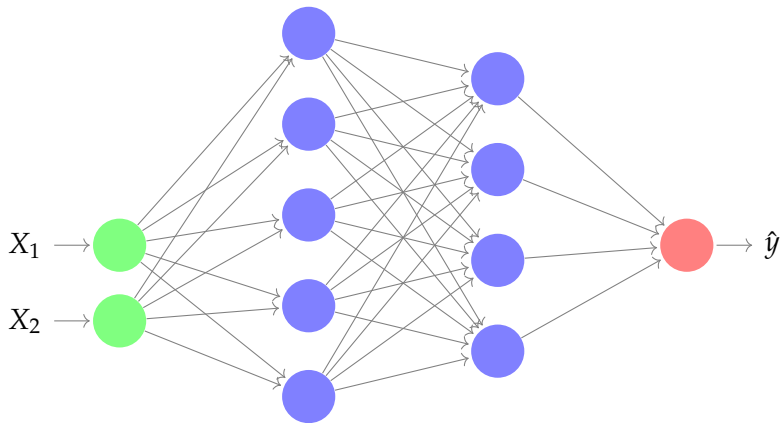
Architecture Design
Network Tuning

- ▶ 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

  - ▶ Details of stochastic gradient descent.

  - ▶ Regularization of parameters

- ▶ This is an active research area that involves a lot of trial and error, and overfitting is a latent danger at each step.

Training the network
Example: MNIST

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?

When to Use Deep Learning?