# Introduction to Deep Neural Networks

Máster Universitario en Ciencia de Datos - Métodos Avanzados en Aprendizaje Automático

Carlos María Alaíz Gudín

Escuela Politécnica Superior
Universidad Autónoma de Madrid

Academic Year 2020/21

UAM
Universidad Autónoma
de Madrid

# Contents

# Introduction

# DNN Origins

1. MLPs (NNs with only one-hidden layer) were the state-of-the-art models during the 80s and 90s.
2. Due to the **universal approximation property** of the MLPs, deeper networks were not considered.
3. With the apparition of Kernel Methods in the 90s, their use decreased considerably.

---

4. They resurged again with the **Deep Learning** paradigm introduced by Hinton in 2006.

**Definition (Deep Learning)**

**Deep Learning** (DL) is a type of machine learning based on artificial neural networks in which multiple layers of processing are used to extract progressively higher level features from data.

# MLPs with Several Hidden Layers: Limitations

1. A main limitation of the MLPs with multiple hidden layers was the **vanishing gradient** problem.
   - The gradient tends to get smaller as it is propagated during the backward phase.
   - As a result, only the last layers are really trained, while the initial ones are kept almost unchanged.
   - It was easier to train an MLP with many hidden units in a single hidden layer than with many layers.

---

2. Another difficulty is just the computational cost of training a large neural network.

**Notebook**

Vanishing Gradient

# MLPs with Several Hidden Layers: Diagnosis

- Hinton summarized why MLPs with several networks used to not work:
  1. *Our labeled datasets were thousands of times too small.*
     $\Rightarrow$ Larger datasets.
  2. *Our computers were millions of times too slow.*
     $\Rightarrow$ More computational power.
  3. *We initialized the weights in a stupid way.*
     $\Rightarrow$ Clever initialization.
  4. *We used the wrong type of non-linearity.*
     $\Rightarrow$ New activation functions.

# Innovations of Deep Learning

# Big Data

- The **Industry 4.0** and the **Digital Transformation** implied a revolution in the management of data.
- The interest of the companies and institutions changed in two phases:
  1. Collecting data and applying new technologies.
  2. Trying to gather information and extracting value from the collected data.

---

- This transformation resulted in the availability of a huge amount of heterogeneous data.
  - **Big data** paradigm.

---

- The machine learning models can (and have to) be trained with much more data than before.

# Computational Power

- The computational power of CPUs has increased consistently, influenced among other by **Moore's law**.
- The number of threads available per CPU has also raised.

---

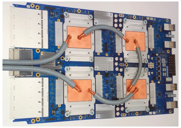
- A key factor in the development of DL is the usage of Graphics Processing Units (GPUs), which can handle hundreds of threads.
- This allow for a huge degree of parallelization in matrix calculus.

---

- Some companies (e.g. Google) has developed specific DL hardware as the Tensor Processing Units (TPUs).

CPU.


GPU.


TPU.

# Initialization

- Deep NN can be properly trained if the weights are correctly initialized.
  - If they are too small, the gradient will vanish.
  - If they are too large, the learning can be very slow.
- There are several heuristics to initialize the weights effectively.

**Xavier Initialization** The weights are initialized using a Gaussian with zero mean and variance $\frac{1}{d_{\ell-1}}$, where $d_{\ell-1}$ is the number of input units to each layer.

**Uniform Initialization** The weights are initialized using a uniform distribution around zero with bounds $\pm\sqrt{2/(d_{\ell+1}+d_\ell)}$ (the constant 2 depends on the activation function).

**Transfer Learning** The weights of a successfully trained model used in a similar problem are used as initial weights.

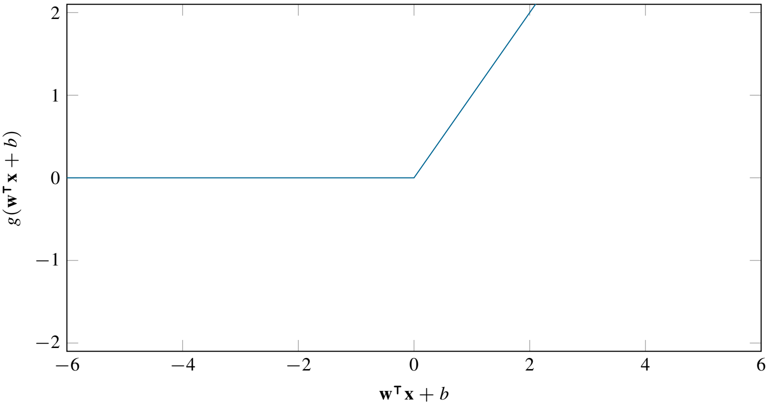
Notebook

Weight Initialization

# Activation Functions (I)

**Rectified Linear (ReLU)** $g(a) = \max\{0, a\}$.

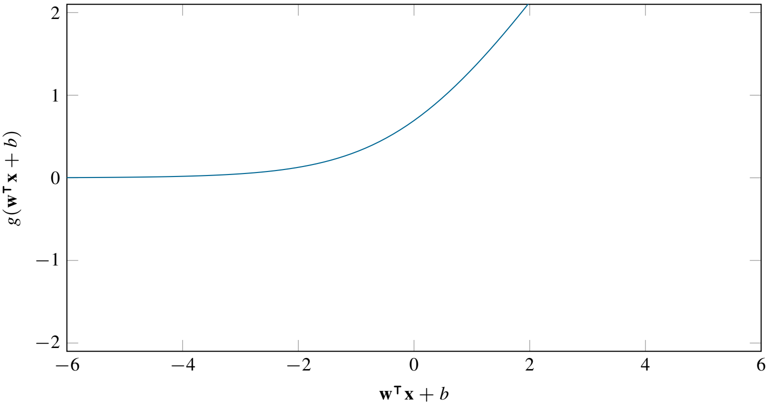
- Sparse, the gradient does not vanish.
- Continuous but non-differentiable at 0.



# Activation Functions (II)

**Softplus** $g(a) = \ln(1 + e^a)$.

- Smooth version of the ReLU.
- Continuous and differentiable, although no-sparse.



Notebook

Activation Functions

# Avoiding Over-Fitting: Data Augmentation

- With the increment in the flexibility of NNs a problem arises: the risk of **over-fitting**.

---

- A large amount of data prevents from over-fitting.
  - Not always available, it depends on the problem.

---

- A first solution is to generate new data (**data augmentation**).
- This process is not trivial, the generated data has to be relevant for the problem.
- Different approaches:
  - Perturbing with noise.
  - Fitting the distribution of the original data.
  - Using expert knowledge about the variations in real-life (particularly useful with images).

Notebook

Data Augmentation

# Avoiding Over-Fitting: Transfer Learning

- The model can be pre-trained in a large dataset, and then adapted to the problem at hand.
- This approach is known as **transfer learning**.

## Transfer Learning

1. Take a model successfully trained over a larger dataset.
   - The complete model, or only a part of it (usually the feature extraction).
2. Add the necessary layers for adapting it to the problem at hand.
3. Train the new layers.
4. Train all the layers with a smaller learning rate (**fine tuning**).

Notebook

Transfer Learning

# Avoiding Over-Fitting: Dropout

- A typical approach to regularize a DNN is the **dropout**.

---

1. During training, a certain percentage $r$ of the inputs to a hidden layer are set to 0, while the remaining inputs are increased as $\frac{1}{1-r}$ to compensate the scale.
   - The network "learns" to distribute the information processing, not relying on single units.
2. During the prediction of new data all the units are considered as usual.

Notebook

Dropout

# Specialized Architectures

- 

# Introduction to Deep Neural Networks

Carlos María Alaíz Gudín