Neural Networks: recent advances - Deep learning - and some (mathematical ?) problems

Rui Rodrigues

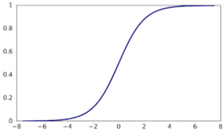
FCT/UNL, CMA

Linear and Logistic Regression

If we have a **finite** set, $Z \subset X \times \mathbf{R}$, of data points from a 'universe', U, the basic choices to model U are

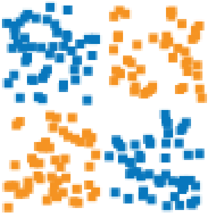
- Linear Regression: $(x, y) : y = x * W + b$
- Logistic Regression (x, y) : $y = \sigma(x * W + b)$

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$



$\sigma(t) \in]0, 1[$, used mostly for binary classification 0/1
(it can be generalized for finite number of classes: softmax)

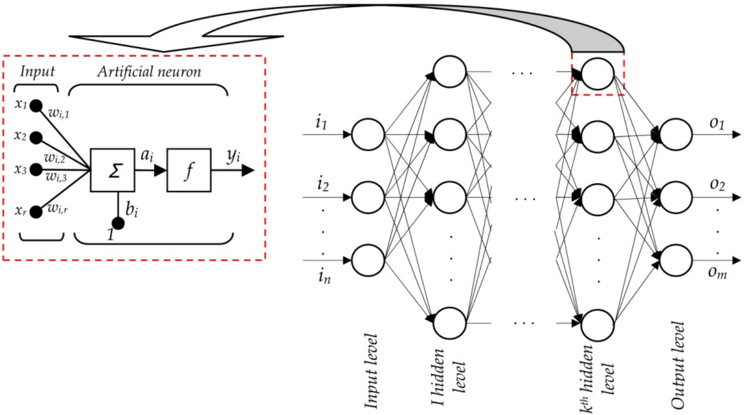
They will fail to model very simple situations:



They can not even model the given **set of examples**.

However they can be used as the elementar blocks of a general model: neuralnetworks

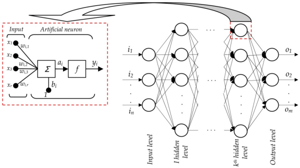
Neuralnetworks



Deep Learning = many hidden layers

Hidden layers can be seen as **representations** or **features** of the input data

Neuralnetworks

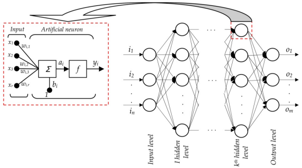


Nowadays neuralnetworks have **thousands of units** on the hidden layers and millions of parameters. These parameters have to be estimated to fit $\approx$ the set of examples.

Main idea: **Gradient Descend** on the error

Successive Approximations

Neuralnetworks



Nowadays neuralnetworks have **thousands of units** on the hidden layers and millions of parameters. These parameters have to be estimated to fit ≈ the set of examples.

Main idea: **Gradient Descend** on the error
Successive Approximations

$$W_{k+1} = W_k - \epsilon \nabla Error$$

Stochastic Gradient Descend

- A bit more concrete: Stochastic Gradient Descend
- We run throught the training set of examples many times (epochs).
- In one epoch, we don't use the whole training set of examples on each step (parameters update):
  the training set is randomly divided in mini-batches(≈ 100?) used for each update.
- Why dont we get stuck on local minima?

Some of the Deep Learning Main Achievements

- Object recognition on images
- Voice recognition (transcription voice to text)
- Voice synthesization (text to speech)
- Language translation
- Deep Reinforcement Learning (winner of 'GO')

How to avoid overfitting

- The intention is not only to fit the model to the training data set of examples but also to generalize to new 'similar' examples.
- A large (complex) model may model the training set but not generalize.
- We use **regularization** to try to avoid overfitting - apply further constrainst on the weights :
  - Early stop the training
  - L^1 or L^2 penalty on the weights.
  - **Dropout: randomly set to zero the output of a percentage of the neuralnetwork units**
  - There exist some other very new regularization methods
  - new regularization methods?

Some variants of the **feedforward** neural network model

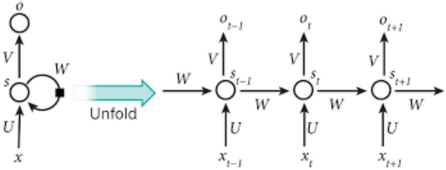
- Convolution neural networks
- Recurrent neural networks
- Generative Models
- Deep Reinforcement Learning

Recurrent neural networks

- They are used to work with sequential data: time series, voice recognition, text understanding and translation

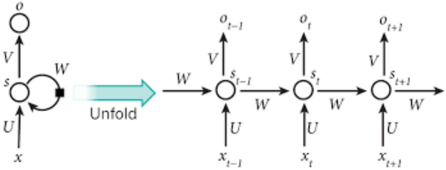
Recurrent neural networks

- They are used to work with sequential data: time series, voice recognition, text understanding and translation
- single hidden layer:

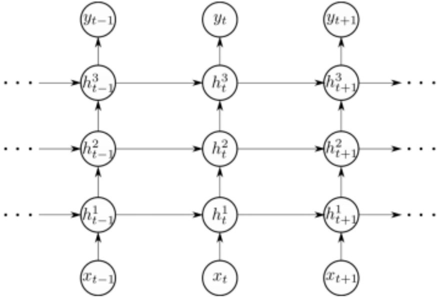


Recurrent neural networks

- They are used to work with sequential data: time series, voice recognition, text understanding and translation
- single hidden layer:



- several hiden layers:



Recurrent neural networks

- Weights coming from previous time steps transmit information from the past
- However W^n may explode or get too near to zero.
  To train recurrent neural networks we need to use some tricks.

Generative Models

- the aim is:
  - to generate examples 'similar' to those in the training set
  - and/or, to assign a probability to 'examples'
- Possible tasks assigned to generative models:
  - produce 'natural images', or images of certain objects: cats, persons,..

Generative Models

- the aim is:
  - to generate examples 'similar' to those in the training set
  - and/or, to assign a probability to 'examples'
- Possible tasks assigned to generative models:
  - produce 'natural images', or images of certain objects: cats, persons,..
  - Fill in missing data in some dataset (time series?). Could be replacing a part of an image.

Generative Models

- the aim is:
  - to generate examples 'similar' to those in the training set
  - and/or, to assign a probability to 'examples'
- Possible tasks assigned to generative models:
  - produce 'natural images', or images of certain objects: cats, persons,..
  - Fill in missing data in some dataset (time series?). Could be replacing a part of an image.
  - Given an image, generate the same image at a higher resolution (with more details)
  - generate text similar to the one written by somebody (Saramago?).

Generative Models

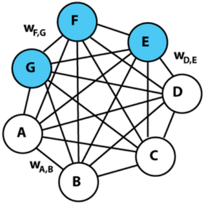
- the aim is:
  - to generate examples 'similar' to those in the training set
  - and/or, to assign a probability to 'examples'
- Possible tasks assigned to generative models:
  - produce 'natural images', or images of certain objects: cats, persons,..
  - Fill in missing data in some dataset (time series?). Could be replacing a part of an image.
  - Given an image, generate the same image at a higher resolution (with more details)
  - generate text similar to the one written by somebody (Saramago?).
  - Given a text, produce the sound of somebody reading that text.
- It is very difficult to evaluate generative models.

Generative Models

There are several types of generative models. We will refer:

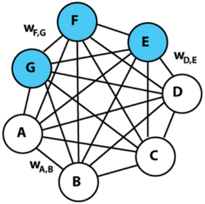
- Boltzmann Machines
- Variational autoencoders
- Generative Adversarial Networks (GANs)

Boltzmann Machines



- It is a network of symmetrically connected stochastic binary units (stochastic version of an **Hopfield Network**)

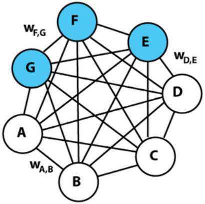
Boltzmann Machines



- It is a network of symmetrically connected stochastic binary units (stochastic version of an **Hopfield Network**)
- The update rule for the value, s_i, of a unit i, depends on its total input
  $z_i = b_i + \sum_j s_j w_{i,j}$, where s_j is the value of a unit j and $w_{i,j}$ is the weight of the connection between units i and j.

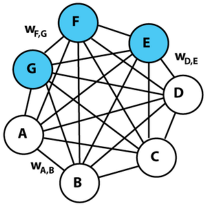
$$probs(s_i = 1) = \frac{1}{1 + e^{-z_i}}$$

Boltzmann Machines



- The probabability of a 'state vector' is
  $P(v) = \frac{e^{-E(v)}}{\sum_u e^{-E(u)}}$
  where
  $E(v) = -\sum s_i^v b_i - \sum s_i^v s_j^v w_{i,j}$
- If the units are updated sequentially the network will eventually reach an equilibrium or stationary distribution (Boltzmann Distribution).

Boltzmann Machines



- Often, the network units are divided in two subsets, visible and hidden units. The values of the visible units might be set to those of the elements of some dataset of binary vectors.
- Given a data set, we can learn values of the weights such that: When the Boltzmann Machine is in equilibrium, it generates on the visible units, with high probability, the elements of the dataset.

Learning Rule for Boltzmann Machines

- Given that $\frac{\partial E(v)}{\partial w_{i,j}} = -s_i^v s_j^v$
  it can be shown that

$$\langle \frac{\partial \log P(v)}{\partial w_{i,j}} \rangle_{data} = \langle s_i s_j \rangle_{data} - \langle s_i s_j \rangle_{model}$$

  where

  - $\langle s_i s_j \rangle_{model}$ is the expected value of $s_i s_j$ when the Boltzmann Machine is sampling state vectors from the equilibrium distribution.
  - When s_i and s_j are visible units, $\langle s_i s_j \rangle_{data}$ is expected value of $s_i s_j$ in the data. When at least one of them is not a visible unit, $\langle s_i s_j \rangle_{data}$ is the average, over all data vectors, of the expected value of $s_i s_j$ when a data vector is clamped on the visible units and the hidden units are repeatedly updated until they reach equilibrium with the clamped data vector.

Learning Rule for Boltzmann Machines

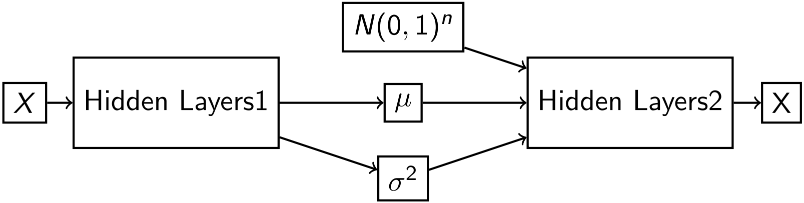
Learning rule:

1. Phase$^+$: Clampe a data vector on the visible units, let the network run to equilibrium, We then increment the weight between any two units which are both on. Repeat a large number of times, with each pattern begin clamped with a frequency corresponding to the the world(=data) probability.
2. Phase$-$: we let the network run freely (no units clamped) and sample the activities of all the units . Once we have reached equilibrium we take enough samples to obtain reliable averages of $s_i s_j$. Then we decrement the weight between any two units which are both on.

Variational Autoencoder

- Generator

- The whole model



Encoder Decoder

Generative Adversarial Networks (GANs)

- Two competing neuralnetworks:

Generative Adversarial Networks (GANs)

- Two competing neuralnetworks:
  1. The generator: creates samples that are intended to come from the same distribution as the training data. The generator uses an input noise 'z' to generate each example.

Generative Adversarial Networks (GANs)

- Two competing neuralnetworks:
  1. The generator: creates samples that are intended to come from the same distribution as the training data. The generator uses an input noise 'z' to generate each example.
  2. The other player is the discriminator. The discriminator examines samples to determine whether they are real or fake(= originate from the generator).
- In the training processes two minibatches are sampled : a minibatch of x values from the dataset and a minibatch generated from a corresponding set of z values.

Generative Adversarial Networks (GANs)

The parameters of each of the networks are updated in the following way:

1. The correction of the generator's weights are associated to the errors of this player: they are the correct answers of the discriminator on the generated samples.
2. The correction of the discriminator's weights are associated to the errors on both the dataset and the generated examples.