Constraints on local primordial non-Gaussianity using a neural network classifier

Biuse Casaponsa Galí
Observational cosmology and instrumentation group
IFCA, Santander

Outline

Constraints on local non-Gaussianity using a neural network classifier

Casaponsa B., Bridges M., Curto A., Barreiro R.B., Hobson M.P., Martínez-González E.

2011, MNRAS,Vol 416,pp 457-464

arxiv:1105.6116

1. Introduction
   - local fnl
   - aim of the work
2. Neural networks
   - General comments
   - Neural network classifier
3. Results

Local non-Gaussianity

Gaussianity

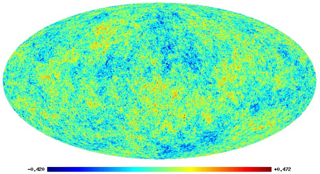
Standard inflationary model $\mapsto$ Gaussian distribution of the anisotropies

Non-Gaussianity

Any deviation from normal probability distribution. Different processes can show different deviations.

local fnl parameter

$\phi = \phi_L + f_{NL}\left[\phi_L^2 - < \phi_L^2 >\right] \Rightarrow \frac{\Delta T}{T} = F(\phi, f_{NL}).$
Third order moments, as for example the bispectrum, are linearly proportional to fnl.



Local non-Gaussianity

Gaussianity

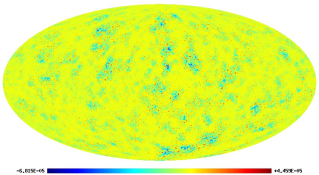
Standard inflationary model $\mapsto$ Gaussian distribution of the anisotropies

Non-Gaussianity

Any deviation from normal probability distribution. Different processes can show different deviations.

local fnl parameter

$\phi = \phi_L + f_{NL}\left[\phi_L^2 - < \phi_L^2 >\right] \Rightarrow \frac{\Delta T}{T} = F(\phi, f_{NL})$.
Third order moments, as for example the bispectrum, are linearly proportional to fnl.



Very weak signal!

Method's efficiency:

- in terms of accuracy = bispectrum (Smith et al. 2009, Komatsu et al.2011, SMHW Curto et al. 2011a,b)
- CPU time (SMHW Curto et al. 2011a,b)

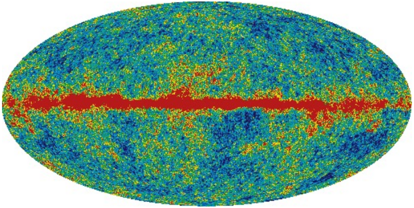
Aim of this work

- We want to show that using neural networks we are able to get equivalent results as the ones obtained through χ^2 minimization, avoiding C estimation and invertion

Estimators

REAL DATA

f_{NL} estimators get more complicated when including the mask and anisoptropic noise.



http://map.gsfc.nasa.gov/

The optimal estimator has been proposed by Creminelli et al. (2006) and succesfully computed by Smith et al. (2009) and Komatsu et al.(2011) for WMAP-5year and WMAP-7year data, giving the best constraints until the moment. $-10 < f_{NL} < 74$.

SMHW analysis using simulations have also found similar results (Curto et al. 2011) $-16 < f_{NL} < 76$.

These methods are computationally demanding. The covariance matrix (non diagonal in the real case) and its inverse has to be estimated.

We want to bypass this last step using neural networks.

Neural networks

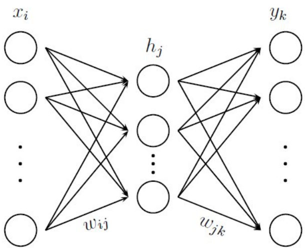


Figure 1. Schematic of a 3-layer feed-forward neural network.

nodes

$$y_k = \sum_j w_{kj} h_j + \theta_k ,$$

where h_j is

$$h_j = tanh(\sum_i w_{ji} x_i) + \theta_j$$

$$y_k = \sum_j w_{kj} \left(\tanh(\sum_i w_{ji} x_i) + \theta_j \right) + \theta_k$$

Supervised training for a feedforward network

We train the network with a known set of inputs and outputs, $\mathbf{x^t}$ and $\mathbf{y^t}$. We choose an optimization function (Ex. mse,rmse,χ^2,...) .

The optimization function is only dependent of the network parameters.

$$Err = \frac{1}{2} \sum_{t,k} (y_k^{(net),t} - y_k^{(t)})^2$$

minimize this function (using conjugates gradient methods, gradient descent method, etc.)

We have used a code developed in Cambridge (Gull and Skilling 1999) with $Q = \alpha S - \chi^2$, where S is the entropy (Gull & Daniell 1978, Skilling 1984). Following the maximum entropy trajectory to find the optimal solution. In any case we need to find w_{lm} and θ_n $\mapsto y_k \sim y_k^{real}$.

Neural Network classifier

$$p_k = \frac{e^{y_k}}{\sum_k e^{y_k}}$$

$$\chi = \sum_k p_k \ln p_k^{(net)}$$

Training

- We need to know how many classes we want and what they represent
- Supervised training requires known $\vec{x}^t$ and $\vec{y}^t$.
- Inputs need to be the characteristic properties of the objects we want to differentiate

Ex. We want to classify apples and oranges. 

1. we need to have a sample of these two fruits, for example 100 apples and oranges.
2. we choose the best properties to differenciate them, ex: color, acidity, texture...
3. we introduce this values to the network.

color (orangeness)	acidity	bumpy texture	Class orange	Class apple
0.9	0.8	0.3	1	0
0.1	0.3	0.01	0	1
0.1	0.8	0.02	0	1
...	...	...	...	...

Neural Network classifier

$$p_k = \frac{e^{y_k}}{\sum_k e^{y_k}}$$

$$\chi = \sum_k p_k \ln p_k^{(net)}$$

Training

- We need to know how many classes we want and what they represent
- Supervised training requires known $\vec{x}^t$ and $\vec{y}^t$.
- Inputs need to be the characteristic properties of the objects we want to differentiate

Ex. We want to classify apples and oranges. 

1. we need to have a sample of these two fruits, for example 100 apples and oranges.
2. we choose the best properties to differenciate them, ex: color, acidity, texture...
3. we introduce this values to the network.

color (orangeness)	acidity	bumpy texture	Class orange	Class apple
0.9	0.8	0.3	1	0
0.1	0.3	0.01	0	1
0.1	0.8	0.02	0	1
...	...	...	...	...

$\vec{x}$=(orangeness,acidity,bumpyness)
$\vec{y}$=(P_{orange},P_{apple})

After the training using N fruits we will get the weights and biases that should be able to generalize the problem. We can put the properties of any orange / apple and the network ouputs will be the probablility of belonging to each class.

In our case we want to know, given a classification of levels of non-Gaussianity, in which level is our data.

1. Sample: Gaussian and non-Gaussian simulated maps with different values of f_{NL}.

$$a_{lm} = a_{lm}^{(G)} + f_{NL} a_{lm}^{(NG)}$$

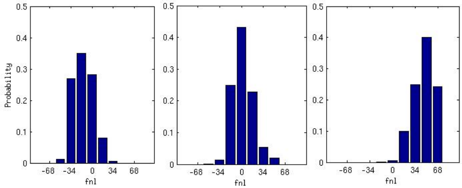
   (Elsner & Wandelt, http://planck.mpa-garching.mpg.de/cmb/fnl-simulations)

2. Characteristic properties: cubic statistics of the wavelet coefficients $S_{jkl} = \sum_i \frac{w_j w_k w_l}{N_{pix}} \mapsto$ 680 inputs

3. Classes: Different levels of non-Gaussianity. (ex. $-100 < f_{NL} <= -80$ class 0, $-80 < f_{NL} <= -60$ class 1,. . .) $\mapsto$ 9 classes

4. After training and testing the network gives the probability of an input vector (statistics of the CMB map) to belong at each class p_i.

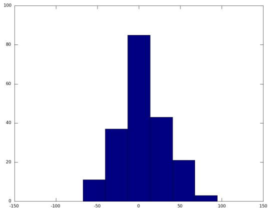
Neural Network classifier



If we do that for 1000 simulations with same fnl, we can compute the bias and dispersion of the estimator $\mapsto$ the efficiency of the method.

We can compute f_{NL} for a given map as:

$$\hat{f}_{NL} = \sum_i fnl_i^{(c)} \times p_i$$



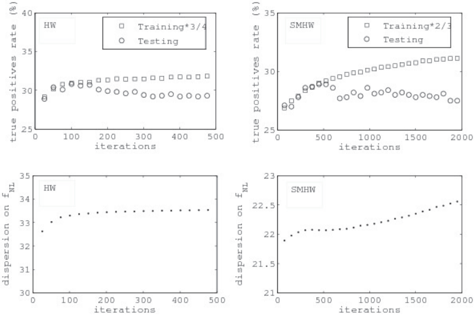
Problems

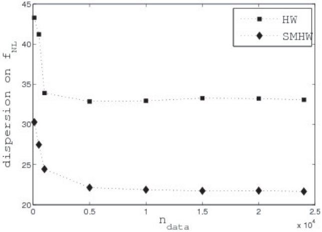
- Working with repeated a_{lm} realizations makes overfitting very likely.
- We detect this when the testing and the training set have divergent behaviour.

Results

Stopping at the moment where the overfitting starts

$$\text{True positives rate} = \frac{\text{Right classified inputs}}{\text{Total of inputs}}$$





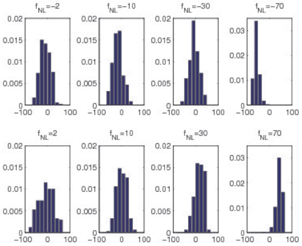
Checking how affects number of hidden nodes and number of inputs to train

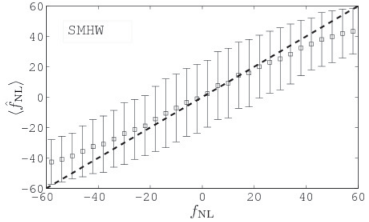
Dispersion computed for $N_{test} = 1000$. $N_{train} = 5000$.

Results

Distribution of $\hat{f}_{NL}$

No significant bias. Edge problem for large f_{NL}





Results

	$\hat{f}_{NLdata}$	$\sigma(\hat{f}_{NL})$	$< \hat{f}_{NLgauss} >$	$P_{2,5}$	$P_{97,5}$
SMHW (NN)	19	22	-1	-43	42
SMHW (WLS) Curto et al. 2011b	32	21	0	-42	46
HW (NN)	-12	33	-1	-66	63
HW (WLS) Casaponsa et al. 2011	6	34	1	-68	67

Results obtained with neural networks (NN) and weighted least squares (WLS). $\hat{f}_{\rm NLdata}$ is the best fitting value for V+W WMAP data, $< \hat{f}_{\rm NLgauss} >$ and $\sigma(\hat{f}_{\rm NL})$ are the expected value and the standard deviation for Gaussian simulations. $P_{2,5}$ and $P_{97,5}$ represent the percentile values at 95 % confidence level of $\hat{f}_{\rm NL}$ for Gaussian realizations.

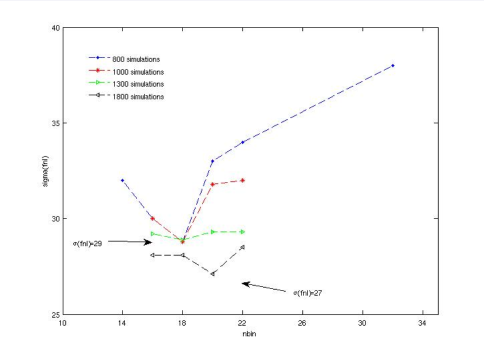
Conclusions

- Neural network estimator of f_{NL} using wavelets coefficients gives same results avoiding the inversion of the covariance matrix.
- Neural network ure seful to solve many to one problems.
- Neural networks might be useful in other cases where matrix invertions are involved.
- We have to be careful with overfitting and network architecture.
- Once the network is trained (in this specific case no more than 1 minute) generalized results are immediate. Point sources, assymetries, etc. can be calculated if simulations available.

Thanks

Results

Binned bispectrum preliminary results



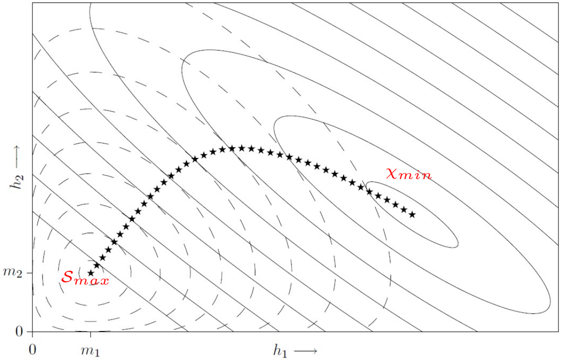
Keep adding simulations to see how it evolves. . .

Training the network

Supervised training for a feedforward network

We train the network with a known set of inputs and outputs, $\mathbf{x^t}$ and $\mathbf{y^t}$. The optimization function is only dependent of the network parameters.

We have used a code based on the MEMSYS package developed by Gull and Skilling 1999 with $Q = \alpha S - \chi^2$, where S is the entropy (Gull & Daniell 1978, Skilling 1984). And followed the maximum entropy trajectory to find the optimal solution. In any case we need to find w_{lm} and $\theta_n \mapsto y_k \sim y_k^{real}$.



Training the network

Supervised training for a feedforward network

We train the network with a known set of inputs and outputs, $\mathbf{x^t}$ and $\mathbf{y^t}$. The optimization function is only dependent of the network parameters.

We have used a code based on the MEMSYS package developed by Gull and Skilling 1999 with $Q = \alpha S - \chi^2$, where S is the entropy (Gull & Daniell 1978, Skilling 1984). And followed the maximum entropy trajectory to find the optimal solution. In any case we need to find w_{lm} and $\theta_n \mapsto y_k \sim y_k^{real}$.