Εθνικο Μετσοβιο Πολυτεχνειο

Δ.Π.Μ.Σ Εφαρμοσμενες Μαθηματικες Επιστημες

# Μπεϋζιανα Νευρωνικα Δικτυα

*Φοιτητής:*
**Νίκος Κουδούνας**

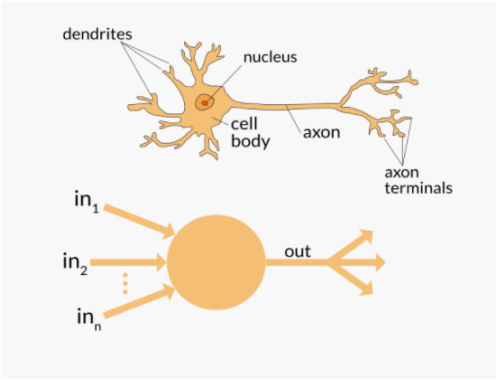
*Καθηγητής:*
**Δημήτρης Φουσκάκης**

Email:**nick.kudunas@gmail.com**

# Περιεχόμενα

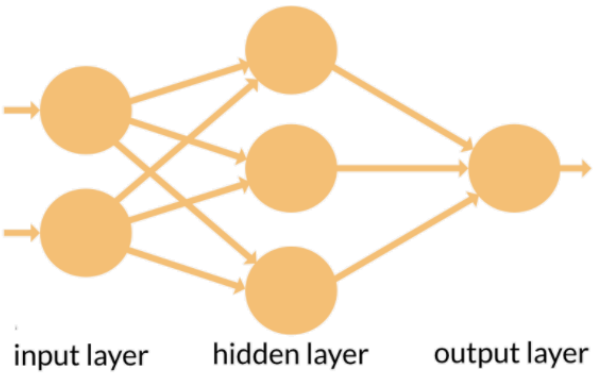
# Εισαγωγή

Μία πολύ καλή ιδέα για να λύσουμε ένα πρόβλημα αναγνώρισης προτύπων είναι να προσπαθήσουμε να μιμηθούμε τον ανθρώπινο εγκέφαλο. Αυτό προκύπτει από την ιδέα ότι ο εγκέφαλος μας είναι το καλύτερο σύστημα το οποίο γνωρίζουμε. Το σύστημα αυτό είναι ένα πολύπλοκο δίκτυο που αποτελείται από $10^{11}$ νευρώνες, ο καθένας από τους οποίους επικοινωνεί με άλλους $10^4$ νευρώνες μέσω ηλεκτρικών σημάτων. Η δομή ενός νευρώνα αποτελείται από ένα κυτταρικό σώμα (που περιλαμβάνει τον πυρήνα και μεγάλο αριθμό οργανιδίων) και από μία η περισσότερες αποφυάδες. Οι αποφυάδες ονομάζονται δενδρίτες όταν συλλέγουν τα σήματα που στέλνονται στο νευρώνα και άξονες όταν τα μεταδίδουν από το κυτταρικό σώμα. Ένα Τεχνητό Νευρωνικό Δίκτυο(Artificial Neural Network) είναι ένα μαθηματικό μοντέλο που προσπαθεί να μιμηθεί το δίκτυο του ανθρώπινου εγκεφάλου. Το δίκτυο αυτό αποτελείται από πολλούς κόμβους (νευρώνες), οι οποίοι παίρνουν κάποια πληροφορία ως είσοδο (π.χ. μία μεταβλητή $\boldsymbol{X} \in \mathbb{R}^m$), μετασχηματίζουν την πληροφορία μέσω κάποιας συνάρτησης ενεργοποίησης και την στέλνουν σε άλλους κόμβους. Όπως ακριβώς δουλεύουν οι δενδρίτες, το κυτταρικό σώμα και οι άξονες σε ένα νευρώνα (Σχήμα 1).




Σχήμα 1: Η δομή ενός πραγματικού και ενός τεχνητού νευρώνα.

Συνδυάζοντας πολλούς νευρώνες μαζί μπορούμε να πάρουμε ένα νευρωνικό δίκτυο όπως στο Σχήμα 2, το οποίο είναι ικανό να λύσει διάφορα προβλήματα αναγνώρισης προτύπων.




Σχήμα 2: Ένα απλό νευρωνικό δίκτυο με δυο εισόδους, ένα κρυφό επίπεδο με τρεις νευρώνες και μία έξοδο.

Σε αυτήν την εργασία θα ασχοληθούμε με νευρωνικά δίκτυα για προβλήματα παλινδρόμησης και κατηγοριοποίησης. Τα μοντέλα αυτά είναι αρκετά σύνθετα και έχουν αρκετές παραμέτρους για εκτίμηση. Από την πλευρά της Μπεϋζιανής Στατιστικής, μπορούμε να θεωρήσουμε αυτές τις παραμέτρους τυχαίες μεταβλητές και να υπολογίσουμε την ύστερη κατανομή τους. Με αυτόν τον τρόπο μπορούμε να ποσοτικοποιήσουμε την αβεβαιότητα στο μοντέλο μας και να έχουμε μία πιο γενική άποψη για τις προβλέψεις του, αλλά και την μεταβλητότητάς τους. Στο πρώτο κεφάλαιο θα δούμε το μαθηματικό μοντέλο που κρύβεται πίσω από την θεωρία των νευρωνικών δικτύων, ενώ στο δεύτερο θα δούμε πώς συνδυάζεται με την Μπεϋζιανή Στατιστική. Ακόμα θα δημιουργήσουμε ένα πιλοτικό πρόβλημα κατηγοριοποίησης και θα τρέξουμε προσομοιώσεις χρησιμοποιώντας την γλώσσα Python 3.5. Τέλος στο τρίτο κεφάλαιο θα δούμε πως προσαρμόζεται ένα τέτοιο μοντέλο και σε ένα πραγματικό πρόβλημα παλινδρόμησης. Το τέταρτο κεφάλαιο είναι το Παράρτημα, όπου υπάρχουν αναλυτικές οδηγίες για την προσομοίωση ενός Μπεϋζιανού μοντέλου στον H/Υ, χρησιμοποιώντας κυρίως την βιβλιοθήκη Μπεϋζιανής Στατιστικής PyMC3 της Python.

# Κεφάλαιο 1

# Μοντελοποίηση ενός προβλήματος

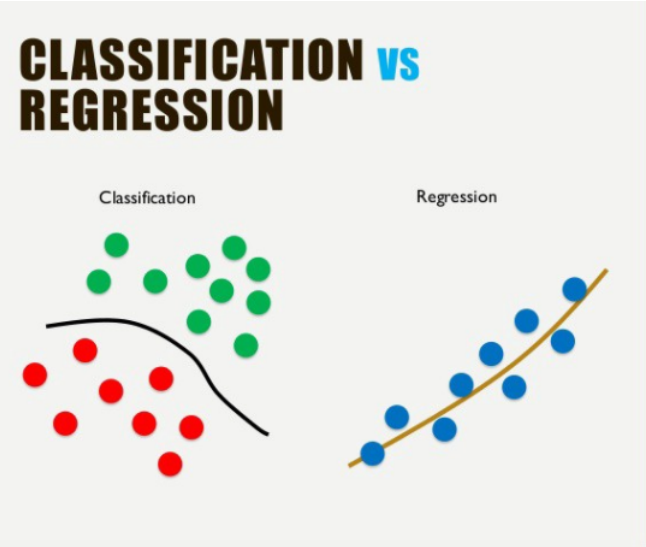
## 1.1 Παραμετρικά και Μη Παραμετρικά Μοντέλα

Στο πρώτο κεφάλαιο θα ξεκινήσουμε κάνοντας μία γενική περιγραφή για το πως μοντελοποιούμε ένα πρόβλημα παλινδρόμησης και κατηγοριοποίησης. Θα δούμε τι υποθέσεις κάνουμε ανάλογα με το μοντέλο που επιλέγουμε και σε τι περιορισμούς καταλήγουμε σύμφωνα με αυτήν την επιλογή. Η γενική μορφή ενός μοντέλου είναι:

$$Y = f(\boldsymbol{X}).$$

Το σκεπτικό πίσω από την παραπάνω σχέση είναι ότι ενδιαφερόμαστε για την τιμή μίας τυχαίας μεταβλητής $Y$, η οποία πιστεύουμε ότι εξηγείται από μία άλλη τυχαία μεταβλητή $\mathbf{X}$. Η βασική υπόθεση που κάνουμε είναι ότι η τ.μ. $Y$ είναι συνάρτηση της $\boldsymbol{X}$. Σε τέτοιου είδους μοντέλα η $Y$ καλείται **μεταβλητή απόκρισης** και η $\mathbf{X}$ **επεξηγηματική μεταβλητή**. Ανάλογα με την μορφή της $Y$ έχουμε διαφορετικό πρόβλημα, αλλά και διαφορετική μοντελοποίηση. Όταν η $Y$ είναι συνεχής, το πρόβλημά μας καλείται πρόβλημα παλινδρόμησης (regression) και ουσιαστικά πρέπει να εκτιμήσουμε την $\mathbb{E}[Y|\boldsymbol{X}]$. Όταν η $Y$ είναι κατηγορική, το πρόβλημά μας καλείται πρόβλημα κατηγοριοποίησης (classification) και πρέπει να εκτιμήσουμε την πιθανότητα $p_j := Pr(Y = j)$, όπου $j$ είναι οι διάφορες κατηγορίες της $Y$. Στο Σχήμα 1.1 βλέπουμε την βασική διαφορά της παλινδρόμησης από την κατηγοριοποίηση. Τα μοντέλα που επιλέγουμε για την ανάλυση αυτών των προβλημάτων χωρίζονται σε δύο κύριες κατηγορίες, τα παραμετρικά και τα μη παραμετρικά μοντέλα. **Παραμετρικά μοντέλα (parametric models)** καλούνται τα μοντέλα στα οποία υποθέτουμε από την αρχή την μορφή της συνάρτησης $f$. Η ονομασία αυτή έχει επικρατήσει με την έννοια

ότι έχουμε επιλέξει μία συγκεκριμένη οικογένεια μοντέλων ως κατάλληλη για την περιγραφή της σχέσης των $\boldsymbol{X}$, $Y$ και το μόνο που πρέπει να κάνουμε είναι να εκτιμήσουμε έναν πεπερασμένο αριθμό από παραμέτρους $\boldsymbol{\theta}$.




Σχήμα 1.1: Αριστερά έχουμε τον διαχωρισμό δύο κλάσεων ($Y = 1$ με πράσινο και $Y = 0$ με κόκκινο) από το σύνορο (μαύρο), όπου $\boldsymbol{X} \in \mathbb{R}^2$. Δεξιά έχουμε δυάδες από σημεία των τ.μ. $(\boldsymbol{X}, Y)$, με $\boldsymbol{X} \in \mathbb{R}$ και την συνάρτηση $f$ που τα συνδέει με καφέ χρώμα.

Εναλλακτικά, θα μπορούσαμε να μην περιοριστούμε σε μία γνωστή οικογένεια και να ψάξουμε λύση σε ένα χώρο με περισσότερα μοντέλα. Μία λύση είναι να χρησιμοποιήσουμε διάφορες μεθόδους για να προσεγγίσουμε την συνάρτηση $f$. Σε αυτές τις περιπτώσεις το μοντέλο θεωρείται **μη παραμετρικό (non parametric)**, με την έννοια ότι δεν κάνουμε καμία υπόθεση ότι ανήκει σε κάποια γνωστή οικογένεια. Ανάλογα την μέθοδο που θα επιλέξουμε για να προσεγγίσουμε την $f$ θα έχουμε και διαφορετική μέθοδο μη παραμετρικής παλινδρόμησης ή κατηγοριοποίησης. Ξεκινώντας από μία σύντομη περιγραφή των βασικών παραμετρικών μοντέλων που έχουμε για την επίλυση προβλημάτων παλινδρόμησης και κατηγοριοποίησης, θα περάσουμε σε μία πολύ γνωστή κατηγορία μη παραμετρικών μοντέλων, την Παλινδρόμηση με συναρτήσεις βάσης. Τέλος θα καταλήξουμε σε ένα διαδεδομένο μοντέλο που ανήκει σε αυτήν την κατηγορία, το νευρωνικό δίκτυο. Θα παρουσιάσουμε το νευρωνικό δίκτυο ως στατιστικό μοντέλο, το οποίο επικαλούμαστε για να λύσουμε κυρίως πολύπλοκα προβλήματα.

### 1.1.1 Παλινδρόμηση

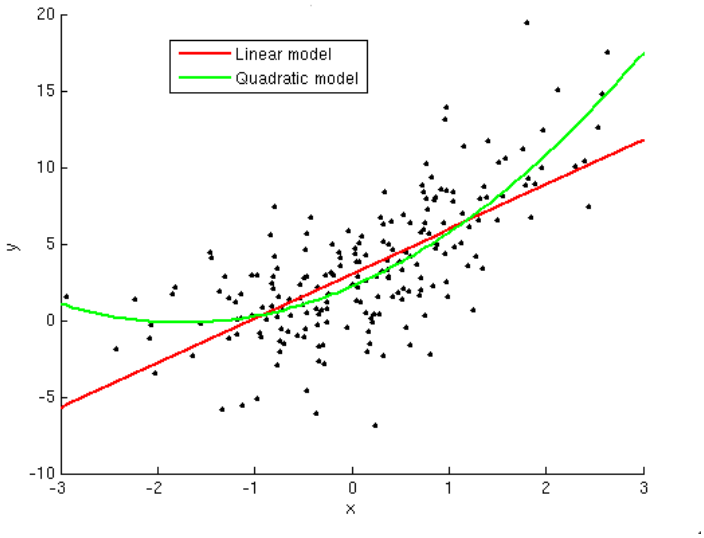
Για αρχή ας υποθέσουμε ότι η τ.μ. $Y$ είναι συνεχής. Όπως είδαμε η βασική μας υπόθεση είναι ότι η $Y$ σχετίζεται με την $X$ μέσω μίας συνάρτησης $f$. Σκοπός μας είναι να δημιουργήσουμε ένα στατιστικό μοντέλο στο οποίο θέλουμε να προσδιορίσουμε την τιμή της τ.μ. $Y|\boldsymbol{X}$, δηλαδή την τιμή της $Y$ αν γνωρίζουμε την τιμή της $\boldsymbol{X}$. Για να περιγράψουμε κατάλληλα το μοντέλο μας θα συλλέξουμε ένα τυχαίο δείγμα προερχόμενο από τον πληθυσμό των $\boldsymbol{X}$ και $Y$. Έστω ότι συλλέξαμε ένα ζεύγος από $N$ παρατηρήσεις $(\boldsymbol{x_i}, y_i)_{i=1}^N$ με $\boldsymbol{x_i} \in \mathbb{R}^m$ και $y_i \in \mathbb{R}$, όπου $m$ είναι ο αριθμός των χαρακτηριστικών της επεξηγηματικής μεταβλητής. Το επόμενο βήμα είναι να διαλέξουμε μια συνάρτηση $\varphi$, η οποία περιγράφει κατάλληλα τα δεδομένα. Η πρώτη σκέψη που μας έρχεται στο μυαλό είναι να χρησιμοποιήσουμε μία γραμμική συνάρτηση. Επειδή το μοντέλο μας είναι στοχαστικό, δηλαδή βασίζεται στην εμπειρική ανάλυσή μας πάνω στο τυχαίο δείγμα, θα προσθέσουμε και έναν όρο $\varepsilon_i$, ο οποίος εκπροσωπεί την τυχαιότητα ( ή το σφάλμα ) του μοντέλου μας και θα πρέπει να προέρχεται από κάποια κατανομή με άγνωστες παραμέτρους. Πολύ απλά, επειδή θέλουμε αυτό το σφάλμα του μοντέλου να είναι κατά μέσο όρο μηδέν, επιλέγουμε μία Κανονική κατανομή με μηδενική μέση τιμή και άγνωστη διασπορά. Σύμφωνα με όλα όσα υποθέσαμε καταλήγουμε στο παρακάτω πολλαπλό γραμμικό μοντέλο ( ή απλό γραμμικό στη περίπτωση όπου $\boldsymbol{x} \in \mathbb{R}$):

$$
\begin{aligned}
&Y_i|\boldsymbol{x_i} \sim N(f(\boldsymbol{x_i}), \sigma^2) \\
&\mathbb{E}[Y_i|\boldsymbol{x_i}] = f(\boldsymbol{x_i}) = \beta_0 + \boldsymbol{\beta^T x_i}.
\end{aligned}
$$

Το $\boldsymbol{x_i} = (x_{i1}, x_{i2}, \ldots, x_{im})$ εκφράζει το διάνυσμα με τις επεξηγηματικές μεταβλητές του μοντέλου για κάθε παρατήρηση. Με $\boldsymbol{\beta} = (\beta_1, \ldots \beta_m)$ συμβολίζουμε το διάνυσμα των συντελεστών και με $\boldsymbol{\theta} = (\beta_0, \boldsymbol{\beta}, \sigma)$ το διάνυσμα των παραμέτρων, το οποίο καλούμαστε να εκτιμήσουμε. Ουσιαστικά η υπόθεση που έχουμε κάνει είναι ότι η τ.μ. $X$ επηρεάζει γραμμικά την αναμενόμενη τιμή της τ.μ. $Y$ δηλαδή ότι $\mathbb{E}[Y|\boldsymbol{X}] = \boldsymbol{b^T X} + b_0$. Πολλές φορές αυτή η γραμμική σχέση δεν είναι αρκετή για να περιγράψει κατάλληλα το μοντέλο. Μία δεύτερη σκέψη θα ήταν να χρησιμοποιήσουμε για $f$ μία πολυωνυμική συνάρτηση ανώτερης τάξης π.χ. ένα πολυώνυμο τάξης 2. Σε αυτήν την περίπτωση το μοντέλο μας θα έχει την παρακάτω μορφή:

$$
Y_i = f(x_i) + \varepsilon_i = b_0 + \sum_{j=1}^{m} b_j x_j + \sum_{j=1}^{m}\sum_{k=1}^{m} c_{ij} x_j x_k + \varepsilon_i, \quad \varepsilon_i \overset{iid}{\sim} N(0, \sigma^2).
$$

Γενικά θα μπορούσαμε να κάνουμε το μοντέλο μας αρκετά πολύπλοκο ανεβάζοντας την τάξη της $f$, όμως θα πρέπει να λάβουμε υπόψιν ότι όσο μεγαλύτερη είναι η διάσταση $m$ της επεξηγηματικής μεταβλητής $\boldsymbol{X}$ τόσο μεγαλύτερος είναι ο αριθμός των παραμέτρων προς εκτίμηση. Επίσης κάνοντας το μοντέλο πιο πολύπλοκο δεν σημαίνει ότι το πρόβλημα που έχουμε να λύσουμε θα περιγράφει καλύτερα τα δεδομένα από το να χρησιμοποιούσαμε ένα πιο απλό μοντέλο. Συνήθως όταν η $\boldsymbol{X}$ είναι μονοδιάστατη και η μορφή του γραφήματος των $\boldsymbol{X}$ και $Y$ μας υποδεικνύει κάποια πολυωνυμική σχέση θα μπορούσαμε να προτιμήσουμε ένα τέτοιο μοντέλο από ένα γραμμικό. Μία τέτοια περίπτωση φαίνεται στο Γράφημα 1.2 όπου ένα πολυωνυμικό μοντέλο ίσως να είναι καταλληλότερο από ένα απλό γραμμικό.




Γράφημα 1.2: Παραπάνω βλέπουμε την συνάρτηση $f$ από ένα γραμμικό μοντέλο(πράσινη ευθεία) και ένα πολυωνιμκό τάξης 2 (κόκκινη γραμμή).

Στο γραμμικό μοντέλο υποθέτουμε ότι σχέση της τ.μ $Y$ με την τ.μ. $\boldsymbol{X}$ είναι γραμμική και τα σφάλματα είναι ανεξάρτητα και ισόνομα $(iid)$ και άρα θέλουμε να εκτιμήσουμε μόνο το $\boldsymbol{\theta} = (\beta_0, \boldsymbol{\beta}, \sigma)$, δηλαδή συνολικά $m+1$ παραμέτρους. Αυτό το μοντέλο καλείται παραμετρικό, αφού επιλέξαμε από την αρχή την μορφή της $f$. Η οικογένεια που διαλέξαμε είναι η κανονική, $Y|\boldsymbol{X} \sim N(\boldsymbol{\beta^T X} + \beta_0, \sigma^2)$ με την μέση της τιμή να προέρχεται από την γραμμική σχέση των $\boldsymbol{X}$ και $Y$. Το πολυωνυμικό μοντέλο είναι και αυτό παραμετρικό μοντέλο σαν το γραμμικό με την μόνη διαφορά ότι η παραπάνω μέση τιμή θα ήταν ένα πολυώνυμο τάξης $d > 1$ της $\boldsymbol{X}$. Η επιλογή της Κανονικής κατανομής έχει να κάνει με την υπόθεση για την Κανονικότητα των σφαλμάτων και όχι με το αν το μοντέλο

είναι παραμετρικό ή όχι.

### 1.1.2 Κατηγοριοποίηση

Έστω ότι αλλάζουμε λίγο την υπόθεσή μας και θεωρούμε ότι η $Y$ είναι δίτιμη τ.μ. με τιμές 0 ή 1. Προφανώς η $Y$ ακολουθεί κατανομή Bernoulli με άγνωστη πιθανότητα επιτυχίας $p$, η οποία όμως εξαρτάται από τις επεξηγηματικές τ.μ. $\boldsymbol{X}$. Σκοπός μας τώρα είναι να μοντελοποιήσουμε την $\mathbb{E}[Y|\boldsymbol{X}] = Pr(Y = 1|\boldsymbol{X})$, δηλαδή την πιθανότητα $p = Pr(Y = 1|\boldsymbol{X})$ ως συνάρτηση της $\boldsymbol{X}$. Το πλέον πιο απλό μοντέλο που χρησιμοποιείται για προβλήματα κατηγοριοποίησης είναι το λογιστικό μοντέλο παλινδρόμησης. Σε αυτό το μοντέλο υποθέτουμε ότι οι δύο κλάσεις ($Y = 0$ και $Y = 1$) είναι γραμμικώς διαχωρίσιμες. Η μορφή του συγκεκριμένου μοντέλου για $N$ στο πλήθος δεδομένα είναι η εξής:

$$Y_i|\boldsymbol{x_i} \sim Bernoulli(p_i)$$

$$ln\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \boldsymbol{\beta^T x_i} = f(\boldsymbol{x_i}),$$

όπου για κάθε $i$ έχουμε ότι $\boldsymbol{\beta},\ \boldsymbol{x_i} \in \mathbb{R}^m,\ \beta_0 \in \mathbb{R}$ και $p_i = Pr(Y_i = 1|\boldsymbol{x_i})$. Η συνάρτηση $f$ που χρησιμοποιείται είναι γραμμική και άρα το μοντέλο είναι παραμετρικό (όπως και στην γραμμική παλινδρόμηση). Η παραπάνω σχέση προκύπτει από την επιλογή της λογιστικής συνάρτησης $\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$ για την μοντελοποίηση της πιθανότητας η τ.μ. $Y$ να πάρει την τιμή 1 ή 0:

$$Pr(Y = 1) = \sigma(\boldsymbol{\beta^T X} + \beta_0) = \frac{e^{\boldsymbol{\beta^T X}+\beta_0}}{1 + e^{\boldsymbol{\beta^T X}+\beta_0}}$$

$$Pr(Y = 0) = 1 - Pr(Y = 1) \Rightarrow$$

$$\frac{Pr(Y = 1)}{1 - Pr(Y = 1)} = e^{\boldsymbol{\beta^T X}+\beta_0} \Rightarrow$$

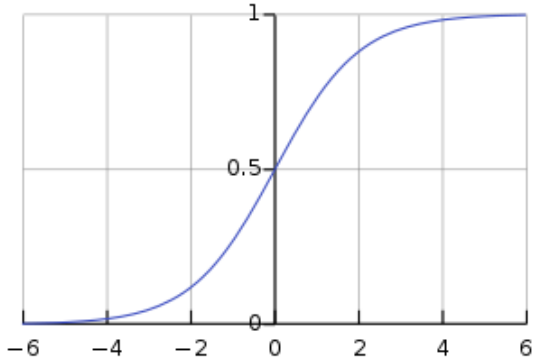
$$ln\left(\frac{Pr(Y = 1)}{1 - Pr(Y = 1)}\right) = \boldsymbol{\beta^T X} + \beta_0 = f(\boldsymbol{X}).$$

Η επιλογή της λογιστικής συνάρτησης έχει γίνει, γιατί έχει δύο πολύ σημαντικές ιδιότητες. Πρώτον μπορεί να μετασχηματίσει την συνάρτηση $f$ έτσι ώστε να έχει πεδίο τιμών το $[0, 1]$, το οποίο είναι αναγκαίο αφού μοντελοποιούμε πιθανότητα

και δεύτερον είναι σιγμοειδής συνάρτηση (S-shape), όπου το σχήμα της μας διευκολύνει στο να βρούμε ένα σύνορο (γραμμικό στην περίπτωσή μας) για να διαχωρίσουμε τις δύο κατηγορίες. Στο Γράφημα 1.3 βλέπουμε το γράφημα της λογιστικής συνάρτησης.

**Ορισμός 1.1.1** *Η συνάρτηση $\sigma : \mathbb{R} \to \mathbb{R}$ ονομάζεται σιγμοειδής αν έχει την παρακάτω ιδιότητα:*

$$\sigma(x) = \begin{cases} 1 & x \to \infty \\ 0 & x \to -\infty. \end{cases} \tag{1.1}$$




Γράφημα 1.3: Από το σχήμα της συνάρτησης μπορούμε να καταλάβουμε ότι αν $\boldsymbol{\beta^T X} + \beta_0 \to \infty$, τότε $p \to 1$. Αντίστοιχα αν $\boldsymbol{\beta^T X} + \beta_0 \to -\infty$ τότε $p \to 0$. Το μοντέλο μας καλείται να προσδιορίσει τις παραμέτρους $\boldsymbol{\beta}, \beta_0$, οι οποίες ξεχωρίζουν την μία κατηγορία από την άλλη.

Ομοίως αν η $Y$ έχει $K \in \mathbb{N}$ κατηγορίες τότε ξέρουμε ότι ακολουθεί κατηγορική κατανομή με διάνυσμα πιθανοτήτων $\boldsymbol{p} = (p_1, \ldots, p_K)$ και $\sum_{k=1}^{K} p_k = 1$. Στην περίπτωση αυτή για να μοντελοποιήσουμε την $p_k = Pr(Y = k) \quad \forall k = 1, \ldots, K$ μπορούμε να χρησιμοποιήσουμε την συνάρτηση $softmax_k(\boldsymbol{z}) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$ $z = (z_1, \ldots, z_K)$, η οποία αποτελεί γενίκευση της λογιστικής συνάρτησης για παραπάνω από δύο κατηγορίες. Σε αυτήν την περίπτωση η πιθανότητα, η τ.μ. $Y|\boldsymbol{X}$ να ανήκει στην κατηγορία $k$ μοντελοποιείται σύμφωνα με τη σχέση $Pr(Y = k) = soft\_max_k(f(X))$, όπου η $f(\boldsymbol{X}) = (f_1(\boldsymbol{X}), \ldots, f_K(\boldsymbol{X}))$ και $f_k(\boldsymbol{X}) = \boldsymbol{\beta_k^T X} + \beta_0$ είναι γραμμική συνάρτηση και άρα

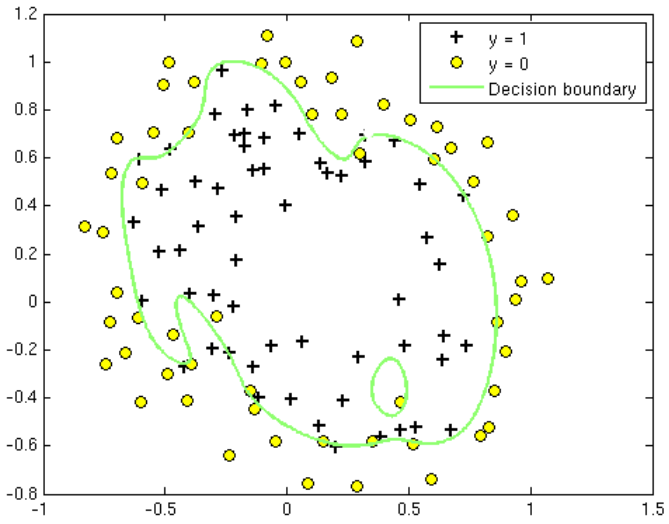
$$Pr(Y = k) = \frac{e^{\boldsymbol{\beta_k^T X} + \beta_{k0}}}{\sum_{j=1}^{K} e^{\boldsymbol{\beta_j^T X} + \beta_{j0}}}.$$

Στην περίπτωση με παραπάνω από μία κατηγορίες θα πρέπει να βρούμε όχι μία, αλλά $K$ στο πλήθος συναρτήσεις $f_k$ για να μοντελοποιήσουμε $K$ στο πλήθος πιθανότητες. Το λογιστικό μοντέλο παλινδρόμησης χρησιμοποιώντας την συνάρτηση $softmax_k()$ με $K$ κατηγορίες και $i = 1, \ldots, N$ είναι το εξής:

$$Y_i|x_i \sim Categorical(p_{i1}, \ldots, p_{iK})$$

$$p_{ik} = \frac{e^{\boldsymbol{\beta_k^T x_i} + \beta_{k0}}}{\sum_{j=1}^{K} e^{\boldsymbol{\beta_j^T x_i} + \beta_{j0}}} \quad \forall \quad k = (1, \ldots, K).$$

Σε αυτό το μοντέλο κατηγοριοποίησης, όπως και στην παλινδρόμηση η επιλογή της συνάρτησης $f$ είναι υποκειμενική. Για παράδειγμα αν οι δύο κλάσεις δεν είναι γραμμικώς διαχωρίσιμες τότε η επιλογή μίας πολυωνυμικής συνάρτησης θα ήταν καταλληλότερη για να περιγράψει τα δεδομένα. Ένα αντίστοιχο παράδειγμα φαίνεται στο Γράφημα 1.4, όπου το πραγματικό σύνορο είναι κυκλικό.




Γράφημα 1.4: Το σύνορο με το πράσινο χρώμα αντιπροσωπεύει την συνάρτηση $f(\boldsymbol{X})$, η οποία είναι πολυωνυμική τάξης 2. Η επεξηγηματική μεταβλητή είναι η $\boldsymbol{X} = (X_1, X_2)$ με την $X_1$ να αντιπροσωπεύει τον οριζόντιο άξονα και την $X_2$ τον κάθετο.

### 1.1.3 Παλινδρόμηση με συναρτήσεις βάσης

Μία αρκετά γνωστή κατηγορία μη παραμετρικής παλινδρόμησης είναι η παλινδρόμηση με συναρτήσεις βάσης (Regression Using Basis Functions). Η ιδέα πίσω από συγκεκριμένο μοντέλο είναι ότι προσπαθούμε να προσεγγίσουμε την συνάρτηση $f$ μέσω άλλων συναρτήσεων, οι οποίες ανήκουν σε ένα σύνολο το οποίο αποτελεί βάση για τον διανυσματικό χώρο που ανήκει η $f$. Συνήθως η $f$ θα είναι στοιχείο του χώρου των συνεχών συναρτήσεων $C(\mathbb{R}^m)$ ή του χώρου των τετραγωνικά ολοκληρώσιμων συναρτήσεων $L^2(\mathbb{R}^m)$. Η θεωρία αυτή πηγάζει από τον κλάδο της Συναρτησιακής ανάλυσης και έχει να κάνει με το ότι κάθε στοιχείο $x$ ενός διανυσματικού χώρου $X$ μπορεί να γραφτεί ως γραμμικός συνδυασμός στοιχείων της βάσης $B$ του $X$. Στην περίπτωση ενός χώρου συναρτήσεων η βάση είναι άπειρο σύνολο (όχι μοναδικό) και άρα ο χώρος είναι απειροδιάστατος. Έτσι για κάθε συνάρτηση $f : \mathbb{R}^m \to \mathbb{R}$ που ανήκει στον $C(\mathbb{R}^m)$, υπάρχουν $\beta_j \in \mathbb{R}$ τ.ω. :

$$f(\boldsymbol{x}) = \sum_{j=0}^{\infty} \beta_j f_j(\boldsymbol{x})$$

όπου $B = \{f_0, f_1, \ldots, f_n, \ldots\}$ μία βάση του $C(\mathbb{R}^m)$. Ένα παράδειγμα βάσης του παραπάνω διανυσματικού χώρου είναι το σύνολο των πολυωνύμων $B = \{f_0(\boldsymbol{x}) = 1, f_1(\boldsymbol{x}) = \boldsymbol{x}, f_2(\boldsymbol{x}) = \boldsymbol{x}^2, \ldots\}$. Πρακτικά για να προσεγγίσουμε αρκετά καλά μία συνάρτηση $f$ χρειαζόμαστε μόνο ένα πεπερασμένο πλήθος στοιχείων της βάσης. Η παλινδρόμηση με συναρτήσεις βάσης προέρχεται από αυτήν την ιδέα. Αν θεωρήσουμε ότι $B$ είναι μία βάση του $C(\mathbb{R}^m)$ ή του $L^2(\mathbb{R}^m)$ τότε το μοντέλο παλινδρόμησης που περιγράφουμε παραπάνω δίνεται από την εξής σχέση:

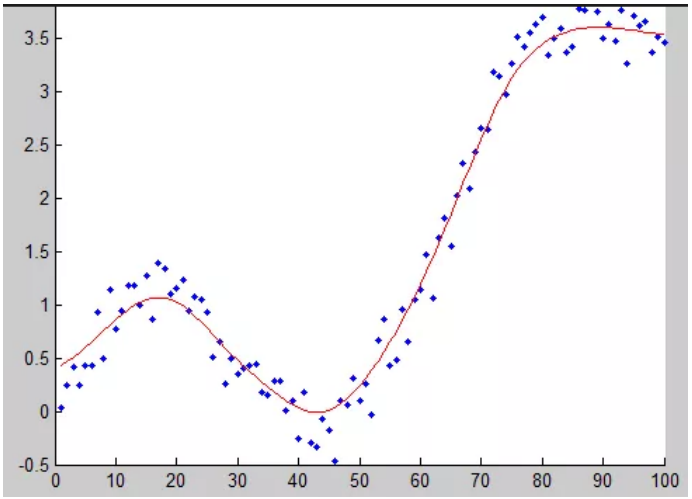
$$\begin{aligned} Y_i|\boldsymbol{x_i} &\sim N(f(\boldsymbol{x_i}), \sigma^2) \\ \mathbb{E}[Y_i|\boldsymbol{x_i}] &= f(\boldsymbol{x_i}) = \sum_{j=0}^{H} \beta_j f_j(\boldsymbol{x_i}). \end{aligned}$$

Συνήθως για $f_0$ χρησιμοποιούμε την σταθερή συνάρτηση $\mathbf{1}$. Άλλα παραδείγματα συναρτήσεων βάσης είναι τα πολυώνυμα Legendre, Laguerre, Hermite, Chebyshev κ.α. Ένα άλλο παράδειγμα που συμπίπτει σε αυτήν την κατηγορία, είναι η παλινδρόμηση με σειρές Fourier στην οποία χρησιμοποιούμε τριγωνομετρικές συναρτήσεις ως βάση για να προσεγγίσουμε την συνάρτηση $f$. Το πιο γνωστό παράδειγμα αυτής της κατηγορίας είναι το μοντέλο μίξης κατανομών. Σε αυτό το μοντέλο χρησιμοποιείται μία συλλογή από Κανονικές κατανομές, των οποίων οι συναρτήσεις πυκνότητας πιθανότητας (σ.π.π) θεωρούνται στοιχεία της βάσης.

Έτσι αν $\Phi(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$ είναι η σ.π.π. της τυποποιημένης Κανονικής κατανομής, τότε το μοντέλο μίξης κατανομών για την παλινδρόμηση στην είναι:

$$Y_i|\boldsymbol{x_i} \sim N(f(\boldsymbol{x_i}), \sigma^2)$$
$$\mathbb{E}[Y_i|\boldsymbol{x_i}] = f(\boldsymbol{x_i}) = \beta_0 + \sum_{j=1}^{H} \beta_j \Phi\left(\frac{\left\|\boldsymbol{x_i} - \boldsymbol{\mu_j}\right\|_2}{\sigma_j}\right).$$

Τα $\boldsymbol{\mu_j}, \sigma_j$ είναι προκαθορισμένοι παράμετροι που πρέπει να αποφασίσουμε πριν ξεκινήσουμε να προσαρμόσουμε το μοντέλο. Τα $\beta_0, \boldsymbol{\beta} = (\beta_1, \ldots, \beta_H)$ ονομάζονται συντελεστές του μοντέλου και είναι οι παράμετροι που πρέπει να βρούμε στο στάδιο της προσαρμογής. Στο Γράφημα 1.5 βλέπουμε ένα παράδειγμα παλινδρόμησης με συναρτήσεις βάσης όπου τα $\sigma_j$ είναι ίδια για κάθε $j$. Το μοντέλο αυτό είναι γνωστό με το όνομα Radial Basis Functions Network (RBFN).

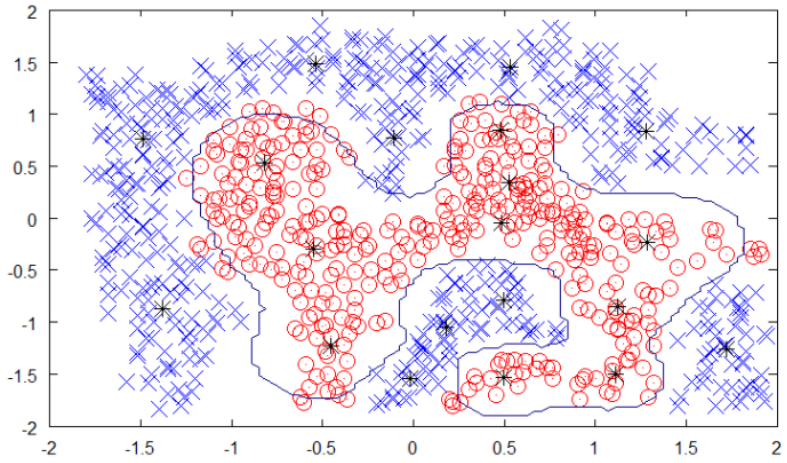



Γράφημα 1.5: Τα σημεία $(x_i, y_i)$ με το μπλε χρώμα είναι τα δεδομένα μας και η $f$ με το κόκκινο χρώμα είναι η συνάρτηση του RBFN που συνδέει τις τ.μ. $X$ και $Y$.

Αυτό το μοντέλο μη παραμετρικής παλινδρόμησης μπορεί να εφαρμοστεί και στην περίπτωση της κατηγοριοποίησης. Το μόνο που αλλάζει, είναι ότι η συνάρτηση $f$ χρησιμοποιείται για να μοντελοποιήσει την πιθανότητα $p$, όπως ακριβώς και στο λογιστικό μοντέλο. Η διαφορά από τα παραμετρικά μοντέλα είναι ότι το μοντέλο αυτό, είναι ικανό να ξεχωρίσει κλάσεις οι οποίες διαχωρίζονται με πολύπλοκο τρόπο όπως στο Γράφημα 1.6.

Το μοντέλο κατηγοριοποίησης με συναρτήσεις βάσης για δύο κλάσεις $Y = 1$ και $Y = 0$ είναι:

$$ln\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \sum_{j=1}^{H} \beta_j \Phi\left(\frac{\|\boldsymbol{x_i} - \boldsymbol{\mu_j}\|_2}{\sigma_j}\right)$$

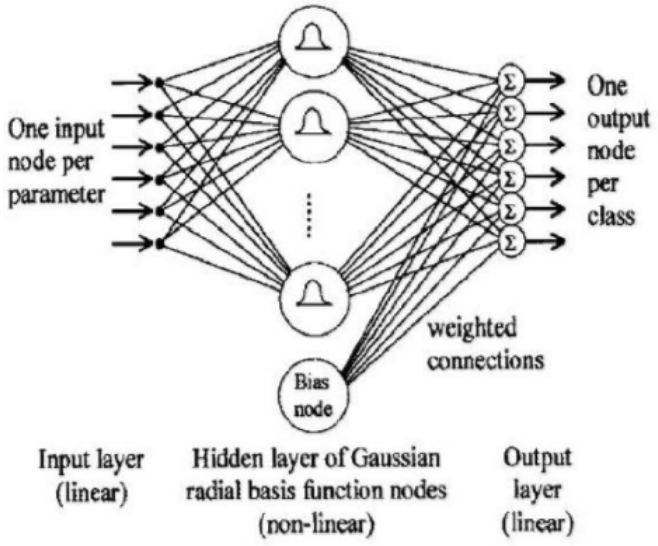



Γράφημα 1.6: Οι δύο κλάσεις $Y = 1$(μπλε) και $Y = 0$(κόκκινο). Το πραγματικό σύνορο είναι αρκετά πολύπλοκο και ένα πολυώνυμο δεν θα ήταν αρκετό για να το περιγραψει. Από την άλλη, η συνάρτηση $f$ του RBFN(μπλε γραμμή) είναι αρκετά κοντά στον να καταφέρει να το προσεγγίσει.

Όλα τα μοντέλα που έχουμε περιγράψει μέχρι τώρα μπορούν να απεικονιστούν με ένα κατευθυνόμενο γράφημα. Ένα κατευθυνόμενο γράφημα είναι ένα διατεταγμένο ζεύγος $G = (V, E)$ όπου $V$, είναι ένα σύνολο του οποίου τα στοιχεία λέγονται κόμβοι και $E$, είναι μια σειρά από διατεταγμένα ζεύγη κόμβων, τα οποία ονομάζονται ακμές ή βέλη. Μία ακμή $e = (x, y)$ θεωρείται ότι κατευθύνεται από τον κόμβο $x$ στον $y$. Ένα κατευθυνόμενο γράφημα ονομάζεται σταθμισμένο, αν ένας αριθμός (βάρος) έχει ανατεθεί σε κάθε ακμή. Οι τιμές των βαρών θα μπορούσαν να αντιπροσωπεύουν, για παράδειγμα, κόστη, μήκη ή ικανότητες, κλπ. ανάλογα με το πρόβλημα κάθε φορά. Στο RBFN για παράδειγμα, το κάθε βάρος αντιστοιχεί στον συντελεστή του κόμβου από τον οποίο προέρχεται η ακμή. Όλα τα μοντέλα που περιγράψαμε μέχρι τώρα μπορούν να απεικονιστούν με ένα σταθμισμένο κατευθυνόμενο γράφημα. Αυτά τα γραφήματα χαρακτηρίζονται από διαφορετικά επίπεδα, στα οποία βρίσκονται οι κόμβοι. Το επίπεδο εισόδου αντιστοιχεί στην τ.μ. $\boldsymbol{X}$ και έχει $m + 1$ κόμβους αν $dim(\boldsymbol{X}) = m$ ( ένας κόμβος αντιστοιχεί στην σταθερά 1 του μοντέλου). Το επίπεδο εξόδου

έχει τόσους κόμβους όσο και η διάσταση της $Y$ στα προβλήματα παλινδρόμησης και έχει $K$ κόμβους σε προβλήματα κατηγοριοποίησης με $K$ κατηγορίες. Αυτά τα δύο επίπεδα ονομάζονται κύρια και υπάρχουν σε όλα τα μοντέλα. Τα πιο σύνθετα μοντέλα έχουν συνήθως και ένα ή παραπάνω κρυφά επίπεδα. Το κρυφό επίπεδο ονομάζεται έτσι, αφού ότι διαδικασία πραγματοποιείται εκεί δεν φαίνεται στο μοντέλο. Ο αριθμός των κόμβων ποικίλει ανάλογα με την περίπτωση.

Αν το πρόβλημα που έχουμε να λύσουμε είναι αρκετά πολύπλοκο ή/και έχουμε αρκετά πολλά δεδομένα, τότε πολλές φορές χρησιμοποιούμε περισσότερα από ένα κρυφά επίπεδα. Αυτά τα μοντέλα ονομάζονται Βαθιά Δίκτυα (Deep Networks). Στο Γράφημα 1.7 βλέπουμε την Αρχιτεκτονική του δικτύου του RBFN για ένα πρόβλημα κατηγοριοποίησης.




Γράφημα 1.7: Στο επίπεδο εισόδου βρίσκεται η τ.μ. $\boldsymbol{X}$. Το κρυφό επίπεδο έχει $H+1$ κόμβους, όπου ο καθένας αντιστοιχεί στην σ.π.π της τυποποιημένης κανονικής $\Phi_j(x)$ για $j = \{1, \ldots, H\}$ και $\Phi_0(x) = 1$, που δηλώνει την σταθερά του μοντέλου (Bias). Το αποτέλεσμα του μοντέλου φαίνεται στο επίπεδο εξόδου και είναι το σταθμισμένο άθροισμα των $\Phi_j$ για κάθε ξεχωριστή κατηγορία.

Το νευρωνικό δίκτυο είναι ένα μοντέλο που ανήκει στην κατηγορία της Παλινδρόμησης/Κατηγοριοποίησης με συναρτήσεις βάσης. Σε αυτήν την περίπτωση η $f$ προσεγγίζεται από ένα πεπερασμένο άθροισμα συναρτήσεων $f_j$ όπου $f_0 = 1$ και $f_j(\boldsymbol{x}) = \sigma(\boldsymbol{\gamma_j^T x} + \gamma_{j0})$ και $j = 1, \ldots, H$. Η συνάρτηση $\sigma$ μπορεί να είναι οποιαδήποτε συνεχής σιγμοειδής συνάρτηση. Η διαφορά με το RBFN, εκτός από τις διαφορετικές συναρτήσεις $f_j$ είναι και ότι έχουμε τα βάρη $\gamma_j, \gamma_{j0}$

που αντιστοιχούν στο επίπεδο εισόδου. Στο επόμενο κεφάλαιο παρουσιάζουμε το θεώρημα που αποδεικνύει ότι ένα απλό νευρωνικό δίκτυο, δηλαδή με μόνο ένα κρυφό επίπεδο, μπορεί να χρησιμοποιηθεί για να λύσει προβλήματα παλινδρόμησης και κατηγοριοποίησης προσεγγίζοντας την συνάρτηση που συνδέει τα δεδομένα.

### 1.1.4 Το Θεώρημα του Cybenko

Το 1989 ο George Cybenko απέδειξε ότι κάθε συνεχής συνάρτηση $f : \mathbb{R}^m \to \mathbb{R}$ με $m \in \mathbb{N}$ μπορεί να προσεγγιστεί αρκετά καλά από ένα νευρωνικό δίκτυο της μορφής:

$$N(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{H} \beta_j \sigma(\boldsymbol{\gamma_j^T x} + \gamma_{j0}),$$

όπου $\boldsymbol{\gamma_j}, \boldsymbol{x} \in \mathbb{R}^m$ και $\beta_j, \gamma_{j0} \in \mathbb{R} \quad \forall j = 1, \ldots, H$, $H \in \mathbb{N}$ και $\sigma(\bullet)$ μία συνεχής σιγμοειδής συνάρτηση. Ουσιαστικά αυτό που απέδειξε ο Cybenko είναι ότι το σύνολο:

$$NN = \{N \in C(\mathbb{R}^n) : N(\boldsymbol{x}) = \beta_0 + \sum_{j=1}^{k} \beta_j \sigma(\boldsymbol{\gamma_j^T x} + \gamma_{j0})\}$$

είναι πυκνό στον $C(\mathbb{R}^m)$ ως προς τη νόρμα $\|f\| = sup\{|f(\boldsymbol{x})| : \boldsymbol{x} \in \mathbb{R}^m\}$ και άρα (σύμφωνα με τα γνωστά αποτελέσματα της Ανάλυσης για πυκνά υποσύνολα ενός διανυσματικού χώρου) για κάθε στοιχείο $f \in C(\mathbb{R}^m)$, υπάρχει ακολουθία συναρτήσεων $\{N_n\} \subseteq NN$ τέτοια ώστε $\|f - N_n\| \to 0$. Αυτή η πρόταση είναι γνωστή ως Universal Approximation Theorem. Η απόδειξη πάει ως εξής:

Έστω $C(\mathbb{R}^m)$ ο διανυσματικός χώρος των συνεχών συναρτήσεων με πεδίο ορισμού τον $\mathbb{R}^m$ για κάποιο $m \in \mathbb{N}$. Όπως είναι προφανές ο $NN$ που ορίσαμε παραπάνω είναι γνήσιος διανυσματικός υπόχωρος του $C(\mathbb{R}^m)$ αφού για κάθε $f, g \in NN \Rightarrow \lambda f + g \in NN$ για κάθε $\lambda \in \mathbb{R}$. Θα δείξουμε ότι ο $NN$ είναι πυκνό υποσύνολο του $C(\mathbb{R}^m)$ με απαγωγή σε άτοπο. Σύμφωνα με τον ορισμό του πυκνού υποσυνόλου, το $NN$ είναι πυκνό στον $C(\mathbb{R}^m)$ ανν $\overline{NN} = C(\mathbb{R}^m)$, δηλαδή η κλειστότητα του $NN$ (το ελάχιστο κλειστό υποσύνολο όλου του χώρου που περιέχει το $NN$) είναι όλος ο χώρος. Έστω ότι $\overline{NN} \neq C(\mathbb{R}^m)$, δηλαδή ο $\overline{NN}$ είναι γνήσιος υπόχωρος του $C(\mathbb{R}^m)$.

**Ορισμός 1.1.2** *Έστω $M(\mathbb{R}^m)$ ο χώρος των πεπερασμένων και κανονικών μέτρων Borel, ορισμένος στον $\mathbb{R}^m$. Κάθε συνάρτηση $\sigma$ καλείται 'διακριτική' (discriminatory) αν για κάθε μέτρο $\mu \in M(\mathbb{R}^m)$ με*

$$\int_{\mathbb{R}^m} \sigma(\boldsymbol{\gamma_j^T x} + \gamma_{j0}) d\mu(\boldsymbol{x}) = 0$$

*και* $\boldsymbol{\gamma_j} \in \mathbb{R}^m, \gamma_{j0} \in \mathbb{R}$ *συνεπάγεται αναγκαστικά ότι* $\mu = 0$

Ο παραπάνω ορισμός είναι πολύ σημαντικός για να αποδείξουμε το Θεώρημα. Αφού καταλήξουμε σε άτοπο θα δείξουμε ότι κάθε συνεχής σιγμοειδής συνάρτηση είναι διακριτική για να ολοκληρώσουμε την απόδειξη. Συνεχίζουμε με δύο πολύ βασικά θεωρήματα της Συναρτησιακής Ανάλυσης.

**Θεώρημα 1** *Θεώρημα Hanh-Banach: Έστω* $X$ *ένας πραγματικός διανυσματικός χώρος και* $p$ *ένα υπογραμμικό συναρτησοειδές ορισμένο στον* $X$*. Αν* $Y$ *υπόχωρος του* $X$ *και* $f$ *ένα γραμμικό συναρτησοειδές ορισμένο στον* $Y$ *τ.ω.* $f(x) \leq p(x)$ *για κάθε* $x \in Y$*, τότε υπάρχει ένα γραμμικό συναρτησοειδές* $F$ *ορισμένο στον* $X$ *τ.ω.* $F(x) \leq p(x)$ *για κάθε* $x \in X$ *και* $F|Y = f$*.*

Ο όρος συναρτησοειδές έχει επικρατήσει στην Συναρτησιακή Ανάλυση για συναρτήσεις με πεδίο ορισμού κάποιον διανυσματικό χώρο και πεδίο τιμών το $\mathbb{R}$. Ο όρος υπογραμμικό συναρτησοειδές $f : X \to \mathbb{R}$ σημαίνει ότι:

$$f(\lambda x) = \lambda f(x)$$
$$f(x + y) \leq f(x) + f(y)$$

για κάθε $x, y \in X$ και $\lambda \in \mathbb{R}$. Η παρακάτω πρόταση είναι μία συνέπεια του θεωρήματος Hanh-Banach που θα χρησιμοποιήσουμε για την απόδειξη μας.

**Πρόταση 1** *Έστω* $X$ *διανυσματικός χώρος με νόρμα και* $Y \subset X$ *γνήσιος κλειστός υπόχωρος του* $X$*. Αν* $x_0 \in X \setminus Y$*, ώστε η απόσταση* $d(x_0, Y) > 0$ *(* $d(x_0, Y) := inf\{d(x_0, y) : y \in Y\}$*), τότε υπάρχει* $L$ *φραγμένο γραμμικό συναρτησοειδές με* $\|L\| = 1$*,* $L|Y = 0$ *και* $L(x_0) = d(x_0, Y)$*.*

Απόδειξη: Η απόδειξη υπάρχει στο [2] και [3]

Σύμφωνα με την παραπάνω πρόταση αφού ο $\overline{NN}$ είναι γνήσιος κλειστός υπόχωρος του $C(\mathbb{R}^m)$, τότε υπάρχει ένα φραγμένο γραμμικό συναρτησοειδές $L$ με $L(\overline{NN}) = 0$ αλλά $L \neq 0$, αφού για κάθε $f \notin \overline{NN} \Rightarrow L(f) = d(f, \overline{NN}) > 0$. Άρα το συναρτησοειδές $L$ είναι μηδεν σε όλο τον υπόχωρο, αλλά θετικό έξω από αυτόν. Αφού $NN \subseteq \overline{NN}$, συνεπάγεται ότι $L(NN) = 0$.

**Θεώρημα 2** *Θεώρημα Αναπαράστασης του Riesz: Έστω* $X$ *ένας τοπικά συμπαγής χώρος Hausdorff. Τότε για κάθε θετικό γραμμικό συναρτησοειδές* $\phi$ *ορισμένο στον* $C_c(X)$*, υπάρχει μοναδικό κανονικό μέτρο Borel* $\mu$ *στον* $X$ *τ.ω.*

$$\phi(f) = \int_X f(x) d\mu(x)$$

*για κάθε* $f \in C_c(X)$*.*

Στην περίπτωσή μας ο $\mathbb{R}^m$ είναι ένας (τοπικά) συμπαγής χώρος Hausdorff (ή T2) και ο $C(\mathbb{R}^m) = C_c(\mathbb{R}^m)$ αφού ο χώρος των πραγματικών συνεχών συναρτήσεων έχει συμπαγές στήριγμα.

Εφαρμόζοντας το παραπάνω θεώρημα στο θετικό γραμμικό συναρτησοειδές $L$, μπορούμε να το γράψουμε στην ακόλουθη μορφή:

$$L(h) = \int_{\mathbb{R}^m} h(\boldsymbol{x}) d\mu(\boldsymbol{x})$$

για κάθε $h \in C(\mathbb{R}^m)$ και μοναδικό $\mu \in M(\mathbb{R}^m)$. Αφού η παραπάνω σχέση ισχύει για κάθε συνάρτηση στον $C(\mathbb{R}^m)$ και η συνάρτηση $\sigma$ είναι συνεχής τότε από τη γραμμικότητα του $L$ συνεπάγεται ότι:

$$L(\sigma) = \int_{\mathbb{R}^m} \sigma(\boldsymbol{\gamma_j^T x} + \gamma_{j0}) d\mu(\boldsymbol{x}) = 0$$

για κάθε $\boldsymbol{x} \in \mathbb{R}^m$ και άρα

$$\int_{\mathbb{R}^m} \sigma(\boldsymbol{\gamma_j^T x} + \gamma_{j0}) d\mu(\boldsymbol{x}) = 0$$

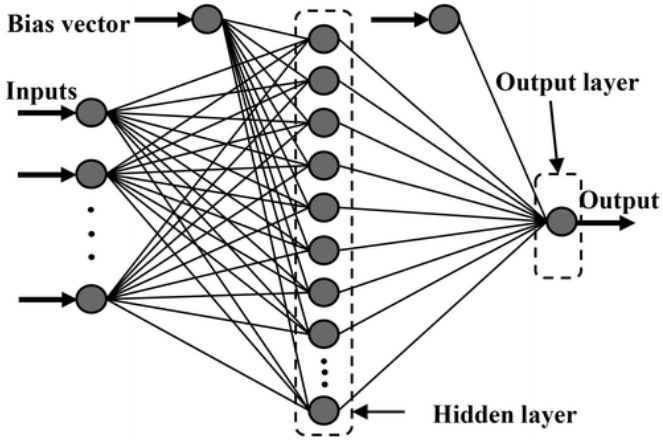
για κάθε $\boldsymbol{\gamma_j, x} \in \mathbb{R}^m$ και $\gamma_{j0} \in \mathbb{R}$. Επειδή η $\sigma$ είναι διακριτική, έχουμε ότι

$$\mu = 0 \Rightarrow \int_{\mathbb{R}^m} h(\boldsymbol{x}) d\mu(\boldsymbol{x}) = 0 \; \forall \; h \in C(\mathbb{R}^m) \Rightarrow L = 0$$

το οποίο έρχεται σε αντίφαση με το ότι ο $L$ είναι μη μηδενικός. Άρα καταλήξαμε σε άτοπο και άρα $\overline{NN} = C(\mathbb{R}^m)$, δηλαδή το σύνολο των νευρωνικών δικτύων είναι πυκνό στον $C(\mathbb{R}^m)$. Αυτό που μένει να δείξουμε ότι ότι κάθε συνεχής σιγμοειδής συνάρτηση είναι διακριτική. Η απόδειξη συνεχίζεται στο Παράρτημα στο τέλος της εργασίας.

### 1.1.5 Το Νευρωνικό Δίκτυο ως Στατιστικό Μοντέλο

Το Θεώρημα του Cybenko αποδεικνύει γιατί ένα νευρωνικό δίκτυο είναι ικανό να προσεγγίσει οποιαδήποτε συνεχή συνάρτηση και άρα και την συνάρτηση $Y = f(\boldsymbol{X})$ του μοντέλου μας. Σε αυτό το μέρος του πρώτου κεφαλαίου παρουσιάζουμε το νευρωνικό δίκτυο ως ένα μη παραμετρικό μοντέλο που χρησιμοποιείται για να λύσει προβλήματα παλινδρόμησης και κατηγοριοποίησης. Η βασική δομή του δικτύου αποτελείται από τρία ¨επίπεδα' (layers). Το επίπεδο εισόδου (input layer), το κρυφό επίπεδο (hidden layer) και το επίπεδο εξόδου (output layer). Οι κόμβοι του γραφήματος ονομάζονται νευρώνες του δικτύου. Στο Γράφημα 1.8 φαίνεται η αρχιτεκτονική ενός απλού νευρωνικού δικτύου.




Γράφημα 1.8: Ένα νευρωνικό δίκτυο με ένα κρυφό επίπεδο. Το Bias δηλώνει την σταθερά 1 του μοντέλου που υπάρχει σε όλα τα επίπεδα εκτός του τελευταίου

Στο πρώτο επίπεδο υπάρχουν τόσοι νευρώνες όσο και η διάσταση της τ.μ. $\boldsymbol{X}$ συν ένας για τη σταθερά του μοντέλου, δηλαδή $m+1$. Οι ακμές από τους νευρώνες του επιπέδου εισόδου προς τους νευρώνες στο κρυφό επίπεδο αντιστοιχούν στα βάρη $\boldsymbol{\gamma} = (\boldsymbol{\gamma_1}, \gamma_{10}, \ldots, \boldsymbol{\gamma_H}, \gamma_{H0})$ τα οποία είναι $(m+1) * H$ στο πλήθος. Έτσι για κάθε $j$ νευρώνα του κρυφού επιπέδου έχουμε $m+1$ βάρη, ώστε η πράξη $\boldsymbol{\gamma_j^T x} + \gamma_{j0}$, όπου $\boldsymbol{\gamma_j} = (\gamma_{j1}, \ldots \gamma_{jm})$ να είναι αποδεκτή για κάθε $j = \{0, \ldots, H\}$. Το αποτέλεσμα της πράξης $\sigma(\boldsymbol{\gamma_j^T x} + \gamma_{j0})$ αντιστοιχεί στον κάθε νευρώνα $j$ του κρυφού επιπέδου, έτσι ώστε η έξοδος του επιπέδου εισόδου

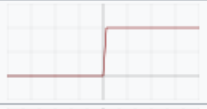
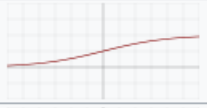
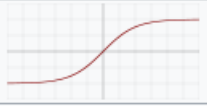
$$h_j(\boldsymbol{x}) := \sigma(\boldsymbol{\gamma_j^T x} + \gamma_{j0})$$

να είναι η είσοδος τους κρυφού επιπέδου. Αντίστοιχα οι ακμές από το κρυφό επίπεδο προς το επίπεδο εξόδου αντιστοιχούν στα βάρη $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_H)$, τα οποία είναι $H+1$ στο πλήθος αφού υπάρχει σταθερά και στο κρυφό επίπεδο. Το αποτέλεσμα της πράξης:

$$o(\boldsymbol{x}) := \beta_0 + \sum_{j=1}^{H} \beta_j h_j(\boldsymbol{x})$$

είναι η έξοδος του κρυφού επιπέδου και αποτελεί την γενική έξοδο του νευρωνικού δικτύου. Η συνάρτηση $\sigma$ ονομάζεται συνάρτηση ενεργοποίησης και σύμφωνα με το θεώρημα του Cybenko, πρέπει να είναι απλά κάποια σιγμοειδής συνάρτηση. Δύο χρόνια αργότερα (1991) ο Hornik απέδειξε ότι η συγκεκριμένη επιλογή της συνάρτησης ενεργοποίησης δεν παίζει και τόσο μεγάλο ρόλο, αλλά η αρχιτεκτονική του δικτύου κάνει την προσέγγιση εφικτή. Συγκεκριμένα

ο Hornik έδειξε ότι μπορούμε να χρησιμοποιήσουμε οποιαδήποτε μη σταθερή, φραγμένη, αύξουσα και συνεχής συνάρτηση $\sigma : \mathbb{R} \to \mathbb{R}$ και το θεώρημα να ισχύει ακόμα. Οι πιο διαδεδομένες συναρτήσεις ενεργοποίησης είναι η δίκτυα συνάρτηση (Binary step), η λογιστική συνάρτηση (logistic), η υπερβολική εφαπτομένη (tanh) και το θετικό μέρος $max\{0, z\}$ ή αλλιώς ReLU(Rectified Linear Unit). Στο Πινακάκι 1.9 φαίνονται οι παραπάνω συναρτήσεις ενεργοποίησης. Η RELU παρόλο που δεν είναι φραγμένη μπορεί να χρησιμοποιηθεί ως συνάρτηση ενεργοποίησης σύμφωνα με το Θεώρημα 2.4 του [4]. Η επιλογή της συνάρτησης ενεργοποίησης είναι πολύ βασική για το μοντέλο και γενικότερα θα πρέπει να θεωρείτε ως παράμετρος που πρέπει να βρούμε πριν την προσαρμογή του.

| | | |
|---|---|---|
| Rectified linear unit (ReLU)[9] |  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ |
| Binary step |  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| Logistic (a.k.a. Soft step) |  | $f(x) = \frac{1}{1+e^{-x}}$ |
| TanH |  | $f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$ |

Πινακάκι 1.9: Παραπάνω βλέπουμε τη μαθηματική εξίσωση και το γράφημα το συναρτήσεων ενεργοποίησης που αναφέραμε.

Σύμφωνα με όσα αναφέραμε παραπάνω, το στατιστικό μοντέλο για ένα νευρωνικό δίκτυο παλινδρόμησης όπου υποθέτουμε ότι τα σφάλματα είναι ανεξάρτητα και ισόνομα με $\varepsilon_i \sim N(0, \sigma^2)$ είναι:

$$
\begin{aligned}
&Y_i|\boldsymbol{x_i} \sim N(f(\boldsymbol{x_i}), \sigma^2) \\
&\mathbb{E}[Y_i|\boldsymbol{x_i}] = f(\boldsymbol{x_i}) = o(\boldsymbol{x_i}) \\
&o(\boldsymbol{x_i}) = \beta_0 + \sum_{j=1}^{H} \beta_j \sigma(\boldsymbol{\gamma_j^T x} + \gamma_{j0}).
\end{aligned}
$$

Αντίστοιχα για ένα πρόβλημα κατηγοριοποίησης το μοντέλο αλλάζει, αφού η έξοδος είναι πιθανότητα και πρέπει να έχει πεδίο τιμών το $[0, 1]$. Η έξοδος $o(\boldsymbol{x})$ περνάει από έναν μη γραμμικό μετασχηματισμό χρησιμοποιώντας την λογιστική

συνάρτηση ή την soft_max. Σην περίπτωση με $K$ κατηγορίες, το μοντέλο λογιστικής παλινδρόμησης είναι

$$
\begin{aligned}
&Y_i|\boldsymbol{x_i} \sim Cat(p_{i1}, \ldots, p_{iK}) \\
&p_{ik} = \frac{e^{o_k(\boldsymbol{x_i})}}{\sum_{j=1}^{K} e^{o_j(\boldsymbol{x_i})}}, \qquad k = 1, \ldots, K \\
&o_k(\boldsymbol{x_i}) = \beta_{0k} + \sum_{j=1}^{H} \beta_{jk} \sigma(\boldsymbol{\gamma_{jk}^T x_i} + \gamma_{j0k}).
\end{aligned}
$$

Το Θεώρημα του Cybenko επεκτείνεται αρκετά εύκολα και σε νευρωνικά δίκτυα με παραπάνω από ένα κρυφά επίπεδα. Το μοντέλο σε αυτήν την περίπτωση γίνεται αρκετά πολύπλοκο, αφού ο μεγαλώνει ο αριθμός των παραμέτρων. Συνήθως, χρησιμοποιούνται όταν έχουμε σύνθετα δεδομένα (πολλές επεξηγηματικές μεταβλητές ή/και μεγάλο όγκο δεδομένων) και άρα η συνάρτηση που θέλουμε να προσεγγίσουμε είναι αρκετά πολύπλοκη, ώστε ένα δίκτυο με ένα κρυφό επίπεδο και πολλούς νευρώνες να μην μπορεί να την βρει. Το δύσκολο με αυτά τα μοντέλα είναι ότι ο σχεδιασμός (κυρίως η επιλογή των παραμέτρων ) πρέπει να γίνει πολύ προσεκτικά και ότι η προσαρμογή του μπορεί να χρειάζεται αρκετά μεγάλο χρόνο, αλλά και υπολογιστικούς πόρους (CPU,RAM,GPU) για να υλοποιηθεί. Γιατί όμως χρειάζεται να σχεδιάσουμε ένα τόσο πολύπλοκο δίκτυο με πάρα πολλές παραμέτρους· Η απάντηση κρύβεται μέσα στο ίδιο το δίκτυο! Αρκεί να καταλάβουμε ότι σε κάθε επίπεδο κάθε ξεχωριστός νευρώνας προσπαθεί να εκπαιδευτεί ώστε να αναγνωρίζει διαφορετικό πρότυπο της συνάρτησης $f$ και άρα όσο πιο πολύπλοκη είναι η μορφή μίας συνάρτησης $f$ τόσο περισσότερα επίπεδα χρειαζόμαστε. Τα νευρωνικά δίκτυα με πάνω από ένα κρυφά επίπεδα ονομάζονται Βαθιά Νευρωνικά Δίκτυα (Deep Neural Networks) και έχουν πολλές εφαρμογές στο πεδίο της Τεχνητής Νοημοσύνης (Artificial Intelligence).

## 1.2 Κλασσική και Μπεϋζιανή Συμπερασματολογία

Η επιλογή ενός μοντέλου πρέπει να γίνεται αρκετά προσεκτικά, λαμβάνοντας υπόψιν όλες τις υποθέσεις που έχουμε κάνει στο στάδιο της μοντελοποίησης. Αφού αποφασίσαμε πιο μοντέλο θα επιλέξουμε, μπορούμε να περάσουμε στο στάδιο της προσαρμογής. Σε αυτό το στάδιο χρησιμοποιούμε τα δεδομένα για να εκτιμήσουμε τις παραμέτρους που περιγράφουν καλύτερα το πρόβλημά μας. Πρακτικά αυτό που πρέπει να κάνουμε είναι να εκτιμήσουμε την παράμετρο $\boldsymbol{\theta}$

της συνάρτησης του μοντέλου $f(\boldsymbol{x};\boldsymbol{\theta})$, λύνοντας ένα πρόβλημα βελτιστοποίησης. Σύμφωνα με τα εργαλεία της Κλασσικής Στατιστικής θα εκτιμήσουμε την πραγματική παράμετρο $\boldsymbol{\theta}$ από την εκτιμήτρια της $\hat{\boldsymbol{\theta}}$. Στην παλινδρόμηση, λόγω της ανεξαρτησίας των σφαλμάτων, η συνάρτηση πιθανοφάνειας $L(\boldsymbol{\theta})$ της τ.μ. $Y|\boldsymbol{X}$ είναι:

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i-\hat{y}_i)^2}{2\sigma^2}}$$

όπου $\hat{y}_i = f(\boldsymbol{x_i};\boldsymbol{\theta})$ είναι η πρόβλεψη του μοντέλου με παράμετρο $\boldsymbol{\theta}$ για κάθε παρατήρηση $i = 1, \ldots, N$. Χρησιμοποιώντας τον λογάριθμο της πιθανοφάνειας έχουμε ότι:

$$\begin{aligned} l(\boldsymbol{\theta}) &= log(L(\boldsymbol{\theta})) \\ &= \frac{-N}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \end{aligned}$$

Σκοπός μας είναι να μεγιστοποιήσουμε την παραπάνω ποσότητα ως προς $\boldsymbol{\theta}$, δηλαδή να βρούμε την Εκτιμήτρια Μέγιστης Πιθανοφάνειας(ΕΜΠ) του μοντέλου. Η μόνη ποσότητα που εξαρτάται από το $\boldsymbol{\theta}$ είναι η πρόβλεψη του μοντέλου $\hat{y}$ και άρα η ποσότητα που πρέπει να μεγιστοποιήσουμε είναι η

$$-\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Αντίστοιχα μπορούμε να ελαχιστοποιήσουμε ως προς $\boldsymbol{\theta}$ την συνάρτηση:

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Η παραπάνω συνάρτηση ονομάζεται συνάρτηση κόστους και ουσιαστικά μας δείχνει πόσο απέχει η πρόβλεψη του μοντέλου μας $\hat{y}_i = f(\boldsymbol{x_i};\boldsymbol{\theta})$ από την πραγματική τιμή $y_i$ σύμφωνα με κάποια μετρική. Εδώ η μετρική που έχουμε χρησιμοποιήσει είναι το Μέσο Τετραγωνικό Σφάλμα -Mean Squared Error(MSE). Από τον ορισμό του προβλήματος βελτιστοποίησης φαίνεται ότι είτε μεγιστοποιήσουμε την πιθανοφάνεια (ΕΜΠ) είτε ελαχιστοποιήσουμε το κόστος (Μέθοδος των ελαχίστων τετραγώνων) θα καταλήξουμε στην ίδια εκτιμήτρια $\hat{\boldsymbol{\theta}}$, αν βέβαια υπάρχει. Στις περιπτώσεις όπου η συνάρτηση κόστους είναι κυρτή, το πρόβλημα λύνεται εύκολα αφού ένα τοπικό ελάχιστο θα πρέπει να είναι ολικό. Η δυσκολία έρχεται στις περιπτώσεις όπου η συνάρτηση κόστους δεν είναι

κυρτή (π.χ. στα νευρωνικά δίκτυα) και άρα υπάρχει περίπτωση να εγκλωβιστούμε σε ένα τοπικό ελάχιστο το οποίο δεν είναι ολικό. Ας περάσουμε τώρα σε ένα πρόβλημα κατηγοριοποίησης με $K$ κατηγορίες. Όπως αναφέραμε παραπάνω, για την μοντελοποίηση των πιθανοτήτων $p_1, \ldots, p_K$ έχουμε χρησιμοποιήσει την συνάρτηση $softmax(k, f)$. Σε αυτήν την περίπτωση, το αποτέλεσμα του μοντέλου για κάθε παρατήρηση $i$ θα είναι ένα διάνυσμα $\hat{\boldsymbol{y}}_{\boldsymbol{i}} = (\hat{y}_{i1}, \ldots, \hat{y}_{iK})$, όπου η κάθε πρόβλεψη αντιστοιχεί στην πιθανότητα η παρατήρηση $i$ να ανήκει στην $k$ κατηγορία. Προφανώς για κάθε $i$ πρέπει $\hat{y}_{i1} + \ldots + \hat{y}_{iK} = 1$. Για λόγους ευκολίας θα αντικαταστήσουμε κάθε παρατήρηση $y_i$ από ένα δυαδικό διάνυσμα $\boldsymbol{y}_{\boldsymbol{i}} = (y_{i1}, \ldots, y_{iK})$ όπου

$$y_{ik} = \begin{cases} 1 & y_i = k \\ 0 & y_i \neq k \end{cases} \tag{1.2}$$

Σύμφωνα με την παραπάνω κωδικοποίηση η συνάρτηση της λογαριθμικής πιθανοφάνειας $l(\boldsymbol{\theta})$ της κατηγορικής τ.μ. $Y|\boldsymbol{X}$ είναι:

$$l(\boldsymbol{\theta}) = \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log \hat{y}_{ik},$$

όπου το $\hat{y}_{ik} = f_k(\boldsymbol{x}_{\boldsymbol{i}}; \boldsymbol{\theta})$ είναι η ποσότητα που εξαρτάται από το $\boldsymbol{\theta}$. Αντί να μεγιστοποιήσουμε την λογαριθμική πιθανοφάνεια μπορούμε να ελαχιστοποιήσουμε την συνάρτηση κόστους $J(\boldsymbol{\theta})$ η οποία είναι γνωστή ως cross entropy:

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log \hat{y}_{ik}.$$

Συνήθως για να αποφύγουμε προβλήματα υπερπροσαρμογής (overfit) ή υποπροσαρμογής (underfit), όπου οι παράμετροι τείνουν να προσαρμόζονται υπερβολικά πάνω στα δεδομένα του μοντέλου ή να γενικεύονται υπερβολικά αντίστοιχα, μπορούμε να προσθέσουμε και έναν όρο ποινικοποίησης στο μοντέλο τ.ω :

$$J_\lambda(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|$$

όπου η $\|\star\|$ συνηθίζεται να είναι η Ευκλείδεια νόρμα ή η νόρμα 1. Διαλέγοντας το $\lambda$ ούτε πάρα πολύ μεγάλο, αλλά ούτε και πάρα πολύ μικρό μπορούμε να έχουμε ένα μοντέλο που γενικεύεται καλύτερα, δηλαδή έχει καλή απόδοση σε νέα δεδομένα στα οποία δεν έχει εκπαιδευτεί. Στην πραγματικότητα, για να εκτιμήσουμε τις παραμέτρους του μοντέλου μπορούμε να χρησιμοποιήσουμε διάφορες αριθμητικές μεθόδους ελαχιστοποίησης, όπως π.χ. Gradient Descent και Stohastic Gradient Descent, ενώ για το $\lambda$ μπορούμε να χρησιμοποιήσουμε κάποια μέθοδο cross validation. Αφού καταφέραμε να υπολογίσουμε την εκτιμήτρια $\hat{\boldsymbol{\theta}}$, μπορούμε να περάσουμε στο στάδιο της πρόβλεψης. Η διαδικασία αυτή

είναι αρκετά εύκολη αφού για κάθε καινούργιο $\boldsymbol{X}^\star$, η πρόβλεψη του μοντέλου μας είναι $Y^\star = f(\boldsymbol{X}^\star; \hat{\boldsymbol{\theta}})$. Χρησιμοποιώντας την μεθοδολογία της Κλασσικής Στατιστικής που περιγράψαμε, καταλήγουμε σε σημειακές εκτιμήσεις για το $\boldsymbol{\theta}$. Αυτό πολλές φορές δεν είναι αρκετό για να περιγράψει την αβεβαιότητα του μοντέλου μας για αυτές τις παραμέτρους, ούτε για τις προβλέψεις του. Από την Μπεϋζιανή σκοπιά μπορούμε να θεωρήσουμε ότι η παράμετρος $\boldsymbol{\theta}$ είναι τυχαία μεταβλητή και άρα ακολουθεί κάποια κατανομή, την οποία θα προσπαθήσουμε να εκτιμήσουμε. Αυτό που αλλάζει τώρα είναι η μοντελοποίηση των παραμέτρων. Πρώτα χρησιμοποιούμε την πεποίθησή μας για το $\boldsymbol{\theta}$, ενσωματώνοντάς την στην λεγόμενη πρότερη κατανομή $Pr(\boldsymbol{\theta})$. Χρησιμοποιώντας τα δεδομένα του προβλήματος, δηλαδή την συνάρτηση πιθανοφάνειας $L(\boldsymbol{\theta}) = Pr(Y|\boldsymbol{X}, \boldsymbol{\theta})$ ανανεώνουμε την πληροφορία μας καταλήγοντας στην ύστερη κατανομή $Pr(\boldsymbol{\theta}|Y, \boldsymbol{X})$ χρησιμοποιώντας τον τύπο του Bayes:

$$Pr(\boldsymbol{\theta}|Y, \boldsymbol{X}) = \frac{Pr(Y, \boldsymbol{X}|\boldsymbol{\theta})Pr(\boldsymbol{\theta})}{\int_{\boldsymbol{\Theta}} Pr(Y, \boldsymbol{X}|\boldsymbol{\theta})Pr(\boldsymbol{\theta})d\boldsymbol{\theta}}$$

Γνωρίζοντας την ύστερη κατανομή του $\boldsymbol{\theta}$, μπορούμε πλέον να εκτιμήσουμε διάφορα χρήσιμα χαρακτηριστικά της όπως την μέση τιμή και τη διασπορά. Βέβαια ο υπολογισμός της ύστερης κατανομής δεν είναι πάντα εφικτός. Το πρόβλημα εμφανίζεται όταν η ύστερη δεν είναι κάποια γνωστή κατανομή που να μπορεί να γραφτεί σε κλειστή μορφή και άρα θα πρέπει να υπολογίσουμε το ολοκλήρωμα του παρανομαστή του τύπου του Bayes που είναι πάνω σε όλο τον παραμετρικό χώρο $\boldsymbol{\Theta}$. Στο επόμενο κεφάλαιο θα δούμε τι εργαλεία έχουμε για να υπολογίσουμε την ύστερη κατανομή των παραμέτρων μας. Εφόσον την υπολογίσουμε, μπορούμε να περάσουμε στο στάδιο της πρόβλεψης. Στην Μπεϋζιανή μοντελοποίηση για κάθε καινούργιο $\boldsymbol{X}^\star$, η πρόβλεψη $Y^\star$ ακολουθεί κάποια κατανομή, η οποία ονομάζεται ύστερη προβλεπτική κατανομή

$$Pr(Y^\star|Y) = \int_{\Theta} Pr(Y^\star|\boldsymbol{X}, \boldsymbol{\theta})Pr(\boldsymbol{\theta}|Y, \boldsymbol{X})d\boldsymbol{\theta}$$

Αφού υπολογίσουμε την παραπάνω κατανομή μπορούμε να πάρουμε κάποιο μέτρο θέσης, π.χ την μέση τιμή της για να δώσουμε μία αντιπροσωπευτική τιμή στην πρόβλεψη $Y^\star$ και κάποιο μέτρο μεταβλητότητας π.χ. την διασπορά της για να δούμε πόσο μεταβάλλεται αυτή η πρόβλεψη.

# Κεφάλαιο 2

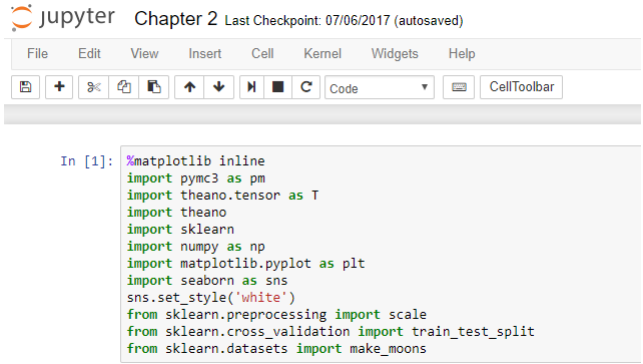
# Μπεϋζιανή Στατιστική και Νευρωνικά Δίκτυα

Στο προηγούμενο κεφάλαιο είδαμε την μαθηματική μοντελοποίηση ενός νευρωνικού δικτύου και πώς αυτό είναι ικανό (σύμφωνα πάντα με την θεωρία) να προσεγγίσει οποιαδήποτε συνάρτηση $f$ περιγράφει τα δεδομένα. Σε αυτό το κεφάλαιο θα δημιουργήσουμε σύνθετα δεδομένα για να δούμε πως δουλεύει στην πράξη ένα Μπεϋζιανό νευρωνικό δίκτυο. Η βασική διαφορά ενός τέτοιου μοντέλου από ένα κλασσικό νευρωνικό δίκτυο είναι ότι θεωρούμε τις παραμέτρους που έχουμε ως τυχαίες μεταβλητές. Σύμφωνα με αυτή την σύμβαση δεν θα βασιστούμε στην ελαχιστοποίηση μίας συνάρτησης κόστους για να εκτιμήσουμε σημειακά αυτές τις παραμέτρους, αλλά θα προσπαθήσουμε με την βοήθεια της Μπεϋζιανής Στατιστικής να εκτιμήσουμε την κατανομή τους. Στην συνέχεια αυτού του κεφαλαίου θα δούμε τι πρότερες κατανομές μπορούμε να δώσουμε στην παράμετρο $\boldsymbol{\theta}$ και πώς υπολογίζουμε την ύστερη κατανομή της

$$Pr(\boldsymbol{\theta}|D) \propto Pr(D, \boldsymbol{\theta})Pr(\boldsymbol{\theta}),$$

όπου με $D = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^{N}$ συμβολίζουμε το δείγμα που έχουμε συλλέξει. Χρησιμοποιώντας την Μπεϋζιανή μοντελοποίηση, ενσωματώνουμε την τυχαιότητα του μοντέλου μας στην ίδια την συνάρτηση $f$, η οποία πλέον είναι τ.μ. αφού έχουμε θεωρήσει ότι η παράμετρος $\boldsymbol{\theta}$ είναι τ.μ. Επίσης θα δούμε πως κάνουμε πρόβλεψη για καινούργια δεδομένα $\boldsymbol{X}^\star$, χρησιμοποιώντας την ύστερη προβλεπόμενη κατανομή $Pr(Y^\star|Y)$ και πως μπορούμε να εισάγουμε την έννοια της αβεβαιότητας στην προβλεπτική ικανότητα του μοντέλου μας εκτιμώντας την τυπική απόκλισή της. Το τελευταίο είναι και το βασικό προτέρημα της Μπεϋζιανής Ανάλυσης, αφού πλέον κάθε καινούργια πρόβλεψη $Y^\star$ δεν θα είναι απλά μία τιμή, αλλά μία τυχαία μεταβλητή με συγκεκριμένη κατανομή. Για την υλοποίηση όλων των παραπάνω θα χρησιμοποιήσουμε την γλώσσα προγραμματισμού Python 3.6. Η επιλογή αυτής της γλώσσας έγινε λόγω δύο συγκεκρι-

μένων βιβλιοθηκών που παρέχει, την Theano και την PYMC3. Η βιβλιοθήκη Theano χρησιμοποιείται για τον υπολογισμό μαθηματικών εκφράσεων. Αυτοί οι υπολογισμοί γίνονται πάνω σε ένα γράφημα με κόμβους, όπου οι κόμβοι πλέον αντιστοιχούν στις μεταβλητές που θα χρησιμοποιήσουμε . Αυτό έχει ως προτέρημα ότι ο υπολογισμός σύνθετων μαθηματικών εκφράσεων γίνεται αποτελεσματικά και γρήγορα χρησιμοποιώντας διάφορα εργαλεία της ¨Θεωρίας Γραφημάτων' (Graph Theory.) Η PYMC3 είναι μία βιβλιοθήκη Πιθανοτικού Προγραμματισμού (Probabilistic Programming), η οποία χρησιμοποιείται για να λύσει προβλήματα Μπεϋζιανής Στατιστικής συμπεριλαμβάνοντας διάφορες μεθόδους για τον υπολογισμό της ύστερης κατανομής. Η PYMC3 είναι βασισμένη πάνω στην Theano και γενικεύει τον τύπο των κόμβων που μπορούμε να χρησιμοποιήσουμε, εισάγοντας την έννοια του στοχαστικού κόμβου. Ουσιαστικά το προτέρημα αυτού είναι ότι ολόκληρη η πληροφορία για την κατανομή μίας τ.μ. δίνεται πλέον από ένα κόμβο στο γράφημα. Συνδυάζοντας όλα τα παραπάνω μπορούμε να κάνουμε την Μπεϋζιανή μοντελοποίηση οποιουδήποτε προβλήματος εφικτή, αν βέβαια μας το επιτρέπει ο εκάστοτε εξοπλισμός μας. Για την παρουσίαση του κώδικα και τον αποτελεσμάτων αυτής της εργασίας θα χρησιμοποιήσουμε ένα εργαλείο της Python που ονομάζεται Jupyter Notebook. Αυτό το εργαλείο είναι ένα Ολοκληρωμένο Περιβάλλον Ανάπτυξης ( Integrated Development Environment (IDE)) που χρησιμοποιείται για πολλές γλώσσες προγραμματισμού στο οποίο τρέχουμε τα παραδείγματα μας, χωρίς να χρειάζεται να χρησιμοποιούμε την κλασσική γραμμή εντολών. Θα ξεκινήσουμε φορτώνοντας στο χώρο εργασίας μας τις βιβλιοθήκες που αναφέραμε, αλλά και κάποιες άλλες που θα χρησιμοποιήσουμε. Πιο γενικές πληροφορίες για όλες τις βιβλιοθήκες και τις συναρτήσεις της Python που δούμε βρίσκονται στο παράρτημα στο τέλος αυτής της εργασίας.

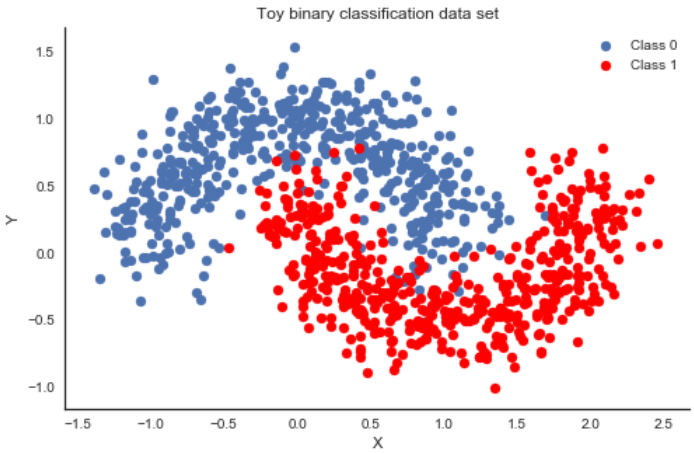



Κώδικας 2.1: Οι πρώτες εντολές για να φορτώσουμε τις βιβλιοθήκες της Python στο Jupyter Notebook

## 2.1 Ένα σύνθετο πρόβλημα

Το πρόβλημα που επιλέξαμε είναι η κατηγοριοποίηση μίας τ.μ. $Y$ με δύο κλάσεις, με επεξηγηματική μεταβλητή (ποσοτική) την $\boldsymbol{X} = (X_1, X_2) \in \mathbb{R}^2$. Δημιουργούμε $N = 1000$ παρατηρήσεις με την συνάρτηση make_moons(), από τις οποίες οι μισές ανήκουν στην κλάση ′0′ και οι άλλες μισές στην κλάση ′1′. Όπως φαίνεται από το όνομα της συνάρτησης, αλλά και από το Γράφημα 2.2 τα δεδομένα του προβλήματός μας σχηματίζουν 2 μισοφέγγαρα, τα οποία είναι οι δύο διαφορετικές κλάσεις που πρέπει να διαχωρίσουμε.

```
X, Y = make_moons(noise=0.2, random_state=0, n_samples=1000)
X = scale(X)
fig, ax = plt.subplots(figsize=(8, 5))
ax.scatter(X[Y==0, 0], X[Y==0, 1], label='Class 0')
ax.scatter(X[Y==1, 0], X[Y==1, 1], color='r', label='Class 1')
sns.despine(); ax.legend()
ax.set(xlabel='X', ylabel='Y', title='Toy binary classification data set');
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.5)
```



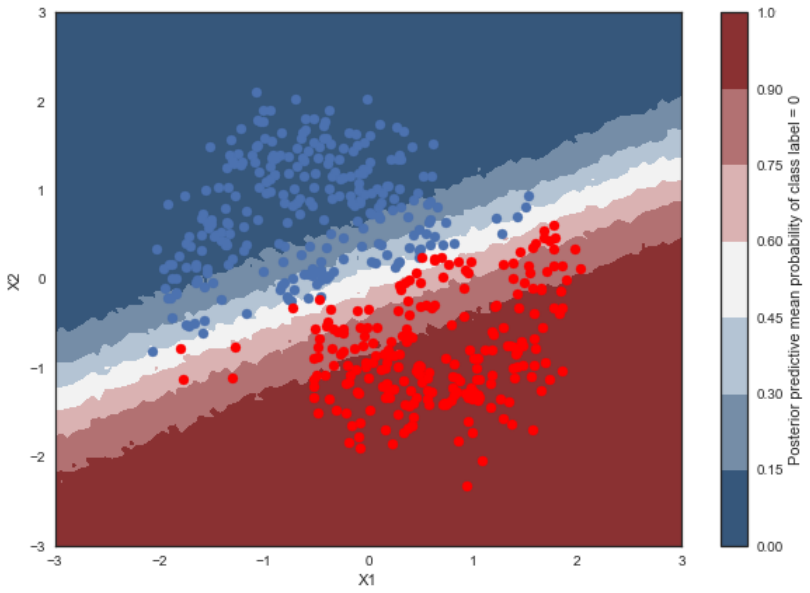

Γράφημα 2.2: Με το μπλε χρώμα είναι τα δεδομένα που ανήκουν στην κλάση 0 και με το κόκκινο αυτά που ανήκουν στην κλάση 1

Πρώτα χρησιμοποιούμε τη συνάρτηση $scale()$ η οποία μετασχηματίζει τα δεδομένα $\boldsymbol{X}$ έτσι ώστε να έχουν μέση τιμή 0 και τυπική απόκλιση 1. Αυτή η κίνηση είναι αρκετά καλή όταν οι μεταβλητές $X_i$ έχουν διαφορετικό εύρος. Με την train_test_split() χωρίζουμε τα δεδομένα μας σε δύο ξένα υποσύνολα του αρχικού, όπου το ένα θα χρησιμοποιηθεί για να εκπαιδεύσουμε το μοντέλο μας (Train_set 50%) και το άλλο για να δούμε πόσο καλά προσαρμόζεται σε νέα δεδομένα (Test_set 50%).

Για αρχή θα μπορούσαμε να διαχωρίσουμε τις δύο κλάσεις χρησιμοποιώντας ένα λογιστικό μοντέλο παλινδρόμησης.

$$log\left(\frac{Pr(Y=1)}{1-Pr(Y=1)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2,$$

όπου $\beta = (\beta_0, \beta_1, \beta_2)$ οι παράμετροι του μοντέλου μας. Υπολογιστικά αυτό το πρόβλημα είναι αρκετά απλό, είτε το μοντελοποιούμε κλασσικά είτε Μπεϋζιανά, αφού η διάσταση της τ.μ. $\boldsymbol{X}$ είναι πολύ μικρή και άρα και η διάσταση του παραμετρικού χώρου $\boldsymbol{\Theta}$. Παρόλα αυτά στο συγκεκριμένο παράδειγμα παραβιάζεται η πιο βασική προϋπόθεση του λογιστικού μοντέλου: οι κλάσεις της $Y$ πρέπει να είναι γραμμικώς διαχωρίσιμες. Στο Γράφημα 2.3 βλέπουμε πως κατανέμεται ο μέσος της ύστερης κατανομής του $p = Pr(Y = 0|\boldsymbol{X})$, όταν το $\boldsymbol{X} \in [-3, 3]^2$, αν χρησιμοποιούσαμε ένα Μπεϋζιανό λογιστικό μοντέλο παλινδρόμησης. Ο κώδικας για αυτό το μοντέλο βρίσκεται στο Παράρτημα.




Γράφημα 2.3: Παραπάνω βλέπουμε την κατηγορία που προέβλεψε το μοντέλο μας για τα δεδομένα ελέγχου, αλλά και πώς κατανέμεται ο μέσος της πιθανότητας επιτυχίας $p$. Η πρότερη που χρησιμοποιήθηκε ήταν μία επίπεδη κατανομή (flat) τ.ω. $log(Pr(\boldsymbol{\theta})) = 0$.

Η λευκή ευθεία θα μπορούσαμε να πούμε ότι είναι το σύνορο των δύο κλάσεων που προέβλεψε το μοντέλο μας. Στην πραγματικότητα το λευκό χρώμα αντιπροσωπεύει το σύνολο των σημείων του $[-3, 3]^2$ στα οποία το μοντέλο προέβλεψε ότι ο μέσος της ύστερης κατανομής της $p$ κυμαίνεται από 0.45 έως 0.60. Ομοίως μπορούμε να ερμηνεύσουμε και τα υπόλοιπα χρώματα ως διαφορετικές μέσες

τιμές της ύστερης κατανομής του $p$. Όπως παρατηρούμε κάποια σημεία είναι σωστά κατηγοριοποιημένα, συγκεκριμένα πάνω από το 80% όπως θα δούμε παρακάτω, όμως αυτό έχει να κάνει με την μορφή του προβλήματος. Το πρόβλημα εδώ είναι ότι ένα λογιστικό γραμμικό μοντέλο δεν είναι ικανό να διαχωρίσει κλάσεις οι οποίες δεν είναι γραμμικώς διαχωρίσιμες και άρα τα διαφορετικού χρώματος σύνορα που βλέπουμε είναι ευθείες γραμμές. Οι προβλεπτικές ικανότητες του μοντέλου μας δίνονται από την συνάρτηση prediction_accuracy (Κώδικας 2.4).

```
pred = prediction_accuracy(trace_LR,logistic_regression,samples=500)
```

```
100%|| 500/500 [00:00<00:00, 1258.40it/s]
Test Accuracy = 85.6%
```

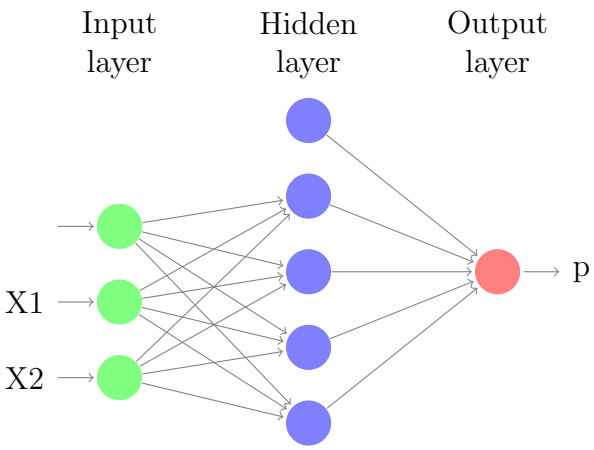
Κώδικας 2.4: Η συνάρτηση prediction_accuracy δημιουργήθηκε στην Python ως ένα μέτρο σύγκρισης για την ικανότητα του μοντέλου να προβλέπει δεδομένα στα οποία δεν έχει εκπαιδευτεί (Test set). Ο κώδικάς της εξηγείται αναλυτικά στο Κεφάλαιο 2.4

Αν η διάσταση της $\boldsymbol{X}$ ήταν μεγαλύτερη, θα ήταν αρκετά δύσκολο έως αδύνατο να καταλάβουμε ότι οι κλάσεις δεν είναι γραμμικώς διαχωρίσιμες και άρα το να χρησιμοποιήσουμε ένα τέτοιο μοντέλο θα ήταν λάθος από την αρχή. Χρησιμοποιώντας ένα νευρωνικό δίκτυο δεν κάνουμε καμία υπόθεση για τον τρόπο με τον οποίο διαχωρίζονται οι κλάσεις μας. Ξεκινάμε με ένα απλό νευρωνικό δίκτυο με ένα κρυφό επίπεδο με πέντε νευρώνες. Η επιλογή του αριθμού των νευρώνων (5), έγινε γιατί τα μοντέλα με πέντε νευρώνες στο κρυφό επίπεδο, είχαν καλύτερη προβλεπτική ικανότητα από τα υπόλοιπα που δοκιμάσαμε (2,4,5,8,10). Οι παράμετροι προς εκτίμηση είναι σύνολο 21, δηλαδή $\boldsymbol{\theta} = (\boldsymbol{\gamma}, \boldsymbol{\beta}) \in \mathbb{R}^{21}$ με $\boldsymbol{\gamma} \in \mathbb{R}^{15}$ τα βάρη από το επίπεδο εισόδου προς το κρυφό και $\boldsymbol{\beta} \in \mathbb{R}^{6}$ τα βάρη από το κρυφό επίπεδο στο επίπεδο εξόδου. Στο κρυφό επίπεδο θα χρησιμοποιήσουμε για συνάρτηση ενεργοποίησης την υπερβολική εφαπτομένη $tanh(x)$, η οποία μετά από δοκιμές που κάναμε, φαίνεται να δουλεύει καλύτερα από την λογιστική συνάρτηση και την RELU. Στο επίπεδο εξόδου θα χρησιμοποιήσουμε την λογιστική συνάρτηση $\sigma(x) = \frac{1}{1+e^x}$ για να μοντελοποίησουμε την πιθανότητα $p$. Το μοντέλο μας για κάθε $i = 1, \ldots, 500$ είναι το ακόλουθο:

$$Y_i | \boldsymbol{x_i} \sim Bernoulli(p_i)$$

$$p_i = \sigma\left(\beta_0 + \sum_{j=1}^{5} \beta_j tanh(\boldsymbol{\gamma_j^T x_i} + \gamma_{j0})\right).$$

.




Σχήμα 2.5: Η αρχιτεκτονική του νευρωνικού δικτύου με πέντε νευρώνες στο κρυφό επίπεδο για το μοντέλο της κατηγοριοποίησης που θα χρησιμοποιήσουμε.

Το νευρωνικό μας δίκτυο καλείται να εκτιμήσει την ύστερη κατανομή της παραπάνω πιθανότητα $p_i$ δεδομένου των τιμών της τ.μ. $\boldsymbol{x_i}$. Η συνάρτηση πιθανοφάνειας $L(\boldsymbol{\theta})$ του μοντέλου μας είναι

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{500} p_i^{y_i}(1-p_i)^{(1-y_i)}.$$

Στην συνέχεια θα πρέπει να αποφασίσουμε ποια θα είναι η πρότερη κατανομή $Pr(\boldsymbol{\theta})$ που θα χρησιμοποιήσουμε για την παράμετρο $\boldsymbol{\theta}$.

## 2.2 Πρότερες Κατανομές

Η πρότερη κατανομή $Pr(\boldsymbol{\theta})$ που χρησιμοποιούμε στα Μπεϋζιανά μοντέλα πρέπει να εκφράζει τις αρχικές μας πεποιθήσεις για την παράμετρο $\boldsymbol{\theta}$. Αυτή η επιλογή είναι καθοριστική για το μοντέλο και θα πρέπει να γίνεται πολύ προσεκτικά. Ας πάρουμε για παράδειγμα ένα γραμμικό μοντέλο παλινδρόμησης με $\boldsymbol{X} \in \mathbb{R}^m$ και παραμέτρους $\boldsymbol{\theta} = (\beta_0, \boldsymbol{\beta} = (\beta_1, \ldots, \beta_m), \sigma) \in \mathbb{R}^{m+2}$. Γνωρίζουμε ότι τα βάρη του μοντέλου $\boldsymbol{\beta}, \beta_0$ είναι συνεχής μεταβλητές με τιμές στον $\mathbb{R}^{m+1}$, ενώ η παράμετρος $\sigma \in \mathbb{R}_+$, αφού είναι αναγκαστικά μη αρνητική. Καταρχάς μία καλή επιλογή για τα $\boldsymbol{\beta}, \beta_0$, θα ήταν μία συνεχής κατανομή με τιμές στον $\mathbb{R}^{m+1}$ και με τέτοιες παραμέτρους που να αντικατοπτρίζουν τις πεποιθήσεις μας για αυτά.

Αν δεν έχουμε κάποια πληροφορία για τα $\boldsymbol{\beta}, \beta_0$, θα μπορούσαμε να χρησιμοποιήσουμε μία αρκετά μη πληροφοριακή πρότερη κατανομή, η οποία θα είναι κεντραρισμένη στο 0 και θα έχει αρκετά μεγάλη διασπορά π.χ. πολυδιάστατα κανονική $N_{m+1}(0, 1e+06I)$ ή πολυδιάστατη ομοιόμορφη $Uniform_{m+1}(1e-06, 1e+06)$. Μία άλλη επιλογή μη πληροφοριακής πρότερης κατανομής είναι η λεγόμενη $flat$, η οποία είναι ανάλογη κάποια σταθεράς $c \in \mathbb{R}$. Το μόνο πρόβλημα με την $flat$ είναι ότι δεν είναι αποδεκτή κατανομή, αλλά κάτω από κάποιες προϋποθέσεις όταν συνδυαστεί με την πιθανοφάνεια μπορεί να δώσει αποδεκτή ύστερη κατανομή. Το προτέρημα αυτής της κατανομής είναι ότι χρησιμοποιείται για να αποφύγουμε προβλήματα υπερχείλισης στον υπολογιστή, τα οποία μπορεί να προκύψουν αν χρησιμοποιήσουμε μη πληροφοριακή κανονική ή ομοιόμορφη κατανομή. Αντίστοιχα αν έχουμε κάποια πληροφορία για τα $\boldsymbol{\beta}, \beta_0$ θα μπορούσαμε διαλέξουμε ως πρότερη κατανομή μία λιγότερο πληροφοριακή κατανομή με ανάλογες παραμέτρους π.χ. την $N_{m+1}(0, 10I)$. Και οι δύο κανονικές κατανομές παρατηρούμε ότι έχουνε μηδενική μέση τιμή. Συνήθως οι μη πληροφοριακές ή λιγότερο πληροφοριακές πρότερες κατανομές που χρησιμοποιούμε είναι κεντραρισμένες στο μηδέν με σκοπό να εκφράσουν ότι η πιθανότερη τιμή για τα $\boldsymbol{\beta}, \beta_0$ είναι μηδέν και άρα η τ.μ. $\boldsymbol{X}$ δεν επηρεάζει την τιμή της $Y$. Επίσης χρησιμοποιώντας τις παραπάνω πρότερες κατανομές (όπου ο πίνακας συνδιασποράς $\Sigma$ είναι διαγώνιος) υποθέτουμε ότι τα $\boldsymbol{\beta}, \beta_0$ είναι ανεξάρτητα και άρα και ασυσχέτιστα. Βέβαια αν τα δεδομένα διαφωνούν με κάποια από αυτές τις αρχικές υποθέσεις, τότε η ύστερη κατανομή που θα προκύψει θα έχει μέση τιμή διαφορετική από μηδεν και $\Sigma$ μη διαγώνιο. Από την άλλη, η παράμετρος $\sigma$ είναι αυστηρά μη αρνητική και άρα θα πρέπει να διαλέξουμε μία συνεχή κατανομή με θετικές τιμές. Συνήθως χρησιμοποιούμε αντίστροφη Γάμμα κατανομή ή περικομμένη στο μηδέν κανονική κατανομή, ώστε $\sigma \in [0, \infty]$. Η λογική που περιγράψαμε παραπάνω εφαρμόζεται σε όλα τα μοντέλα παλινδρόμησης και κατηγοριοποίησης όπου η παράμετρος προς εκτίμηση είναι $\boldsymbol{\theta} = (\boldsymbol{\beta}, \beta_0, \sigma)$ ή $\boldsymbol{\theta} = (\boldsymbol{\beta}, \beta_0)$. Αντίστοιχα και στα νευρωνικά δίκτυα όπου οι παράμετροι είναι $\boldsymbol{\theta} = (\boldsymbol{\beta}, \boldsymbol{\gamma}, \sigma)$ στην παλινδρόμηση ή $\theta = (\boldsymbol{\beta}, \boldsymbol{\gamma})$ στην κατηγοριοποίηση η φιλοσοφία είναι ίδια (σε αυτήν την περίπτωση έχουμε ορίσει τα βάρη $\boldsymbol{\beta}$ και $\boldsymbol{\gamma}$ έτσι ώστε να συμπεριλαμβάνονται οι σταθερές $\gamma_{j0}$ και $\beta_0$ του κάθε επιπέδου στα διανύσματα αυτά). Όπως είπαμε όμως, οι παράμετροι μία πρότερης κατανομής πρέπει να εκφράζουν τις πεποιθήσεις μας για τις παραμέτρους του μοντέλου. Αυτό είναι ένα μεγάλο πρόβλημα για τα βάρη των νευρωνικών δικτύων αφού στις περισσότερες περιπτώσεις δεν έχουνε φυσική ερμηνεία όπως π.χ στο γραμμικό μοντέλο και στο λογιστικό μοντέλο παλινδρόμησης. Έτσι αν έχουμε κάποια αρχική πεποίθηση για τον τρόπο που επηρεάζει η $\boldsymbol{X}$ την $Y$ είναι αρκετά δύσκολο να την ενσωματώσουμε στην πρότερη κατανομή των βαρών. Άρα καταλήγουμε στο ότι μία μη πληροφοριακή πρότερη κατανομή θα ήταν η βέλτιστη επιλογή για τα βάρη ενός νευρωνικού δικτύου. Ας δούμε τι θα μπορούσαμε να κάνουμε στην περίπτωση της κατη-

γοριοποίησης του δικού μας προβλήματος. Οι παράμετροι του νευρωνικού μας δικτύου με ένα κρυφό επίπεδο είναι $\boldsymbol{\theta} = (\boldsymbol{\gamma}, \boldsymbol{\beta}, \sigma)$. Για αρχή θα χρησιμοποιήσουμε μία μη πληροφοριακή πρότερη κατανομή, την flat. Αυτή η κατανομή μοντελοποιείται στον υπολογιστή ως μία συνάρτηση όπου η λογαριθμική κλίμακα αυτής για κάθε τιμή $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ επιστρέφει πάντα την τιμή 0 $(log(Pr(\boldsymbol{\theta})) = 0)$. Το μοντέλο του PMyC3 για το νευρωνικό δίκτυο του πιλοτικού προβλήματος που επιλέξαμε φαίνεται παρακάτω.

```
ann_input = theano.shared(X_train)
ann_output = theano.shared(Y_train)
# Set the number of hidden neurons
n_hidden=5
with pm.Model() as neural_network_flat:
    # Initialize random weights between each layer
    init_1 = np.random.randn(X_train.shape[1], n_hidden)
    init_gamma = np.random.randn(n_hidden)
    init_out = np.random.randn(n_hidden)
    init_beta =np.random.randn()
    # Weights from input to hidden layer
    weights_in_1 = pm.Flat('w_in_1',shape=(X_train.shape[1], n_hidden),
                                                    testval=init_1)
    # Constants from input to hidden layer
    constant_in_1 = pm.Flat('c_in_1',shape=(n_hidden),testval=init_gamma)
    # Weights from hidden layer to output
    weights_1_out = pm.Flat('w_1_out',shape=(n_hidden),testval=init_out)
    # Constants from hidden layer to output
    constant_in_2 = pm.Flat('c_1_out',shape=(),testval=init_beta)
    # Build neural-network using tanh activation function
    act_1 = pm.math.tanh(pm.math.dot(ann_input,weights_in_1)+constant_in_1)

    act_out = pm.math.sigmoid(pm.math.dot(act_1,weights_1_out)+constant_in_2)
    # Binary classification -> Bernoulli likelihood
    out = pm.Bernoulli('out',
                       act_out,
                       observed=ann_output)
```

Κώδικας 2.6: Για λόγους απλότητας αντί να δώσουμε μία πολυδιάστατη κατανομή για όλα τα βάρη του δικτύου $\boldsymbol{\beta}, \boldsymbol{\gamma}$ έχουμε δώσει ξεχωριστές πολυδιάστατες πρότερες κατανομές στις σταθερές $\gamma_{j0},\ j = 1, \ldots, H$ $(constant\_in\_1)$ και $\beta_0$ $(constant\_in\_2)$ του μοντέλου και ξεχωρίστές στα βάρη $\boldsymbol{\gamma_j},\ j = 1, \ldots, H$ $(weights\_in\_1)$ και $\beta_1, \ldots, \beta_H$ $(weights\_in\_2)$.

Στην αρχή ορίζουμε τα δεδομένα εκπαίδευσης $X\_train, Y\_train$ ως theano.shared μεταβλητές, έτσι ώστε να τις αντικαταστήσουμε αργότερα από τις $X\_test, Y\_test$ για να κάνουμε πρόβλεψη στα δεδομένα στα οποία δεν έχει εκπαιδευτεί το μοντέλο μας. Ανοίγουμε το μοντέλο με την εντολή with και με το όνομα $neural\_network$. Οι παράμετροι $init\_0, init\_1$ χρησιμοποιούνται ως αρχικές τιμές για να τρέξει στην αρχή η πρώτη επανάληψη του αλγορίθμου που θα επιλέξουμε για να προσομοιώσουμε τιμές από την ύστερη κατανομή. Στην συνέχεια δίνουμε τις πρότερες κατανομές στα βάρη $\boldsymbol{\gamma}$ και $\boldsymbol{\beta}$ που είναι πολυδιάστατες $flat$ κατανομές με την έτοιμη συνάρτηση $pm.Flat()$. Οι μεταβλητές $act\_1, act\_out$ εκφράζουν τις συναρτήσεις $h(\boldsymbol{x})$ και $\sigma(o(\boldsymbol{x}))$ που είναι η είσοδος του κρυφού επιπέδου και η έξοδος του τελευταίου επιπέδου αντίστοιχα. Τέλος

δίνουμε στην τ.μ. $Y$ την κατανομή που ορίζει η μοντελοποίησή μας, δηλαδή Bernoulli με πιθανότητα $p = \sigma(o(\boldsymbol{x}))$. Αυτό ουσιαστικά είναι το σημείο όπου ορίζουμε την πιθανοφάνεια του προβλήματός μας στο μοντέλο του PYMC3. Αφού έχουμε ορίσει την πιθανοφάνεια και την πρότερη κατανομή, το μοντέλο μας είναι έτοιμο και μένει μόνο να υπολογίσουμε την ύστερη κατανομή. Πριν περάσουμε σε αυτό το στάδιο, θα δούμε ακόμα δύο πρότερες κατανομές που μπορεί να είναι αρκετά χρήσιμες.

Ως πρότερη κατανομή για τα $\boldsymbol{\theta}$ στα νευρωνικά δίκτυα συνηθίζεται να χρησιμοποιείται η αρκετά πληροφοριακή πρότερη κατανομή $N(0, \lambda I_d)$ όπου $d$ είναι η διάσταση του παραμετρικού χώρου $\boldsymbol{\Theta}$. Χρησιμοποιώντας αυτήν την πρότερη κατανομή το πρόβλημα είναι ισοδύναμο με το να προσθέταμε έναν όρο ποινικοποίησης στην συνάρτηση κόστους $J(\boldsymbol{\theta})$ της κλασσικής στατιστικής. Έστω ότι έχουμε ένα πρόβλημα παλινδρόμησης με συνάρτηση $f(\boldsymbol{x}) = o(\boldsymbol{x})$ και συνάρτηση πιθανοφάνειας

$$\prod_{i=1}^{n} N(Y_i | f(\boldsymbol{x_i}), \sigma^2).$$

Αν η πρότερη κατανομή είναι πολυδιάστατη τυποποιημένη κανονική με

$$Pr(\boldsymbol{\theta})) = \prod_{i=1}^{d} N(\theta_i | 0, \lambda) = \frac{1}{\sqrt{2\pi\lambda^2}} e^{-\frac{1}{2\lambda^2}\sum_{i=1}^{d}\theta_i^2},$$

τότε σε λογαριθμική κλίμακα έχουμε ότι:

$$log(Pr(\boldsymbol{\theta})) = -\frac{1}{2\lambda^2}\sum_{i=1}^{d}\theta i^2 + c_1,$$

όπου $c_1$ μία σταθερά που δεν εξαρτάται από το $\boldsymbol{\theta}$. Πολλαπλασιάζοντας την πιθανοφάνεια με την παραπάνω πρότερη έχουμε ότι η ύστερη κατανομή είναι :

$$Pr(\boldsymbol{\theta}|D) \propto \prod_{i=1}^{n} N(Y_i | f(\boldsymbol{x_i}), \sigma^2) \prod_{j=1}^{d} N(\theta_j | 0, \lambda).$$

Δηλαδή σε λογαριθμική κλίμακα η ύστερη κατανομή είναι:

$$log(Pr(\boldsymbol{\theta}|D)) = -\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - f(\boldsymbol{x_i}))^2 - \frac{1}{2\lambda^2}\sum_{i=1}^{d}\theta i^2 + c,$$

όπου $c = c_1 + c_2$ με $c_2$ την σταθερά που προκύπτει όταν λογαριθμήσουμε τον παρονομαστή του τύπου του Bayes.

Θέτουμε $\lambda^\star = \frac{1}{2\lambda^2}$ και αφαιρούμε τη σταθερά $c$ που δεν εξαρτάται από το $\boldsymbol{\theta}$ και άρα

$$log(Pr(\boldsymbol{\theta}|D)) = -\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - f(\boldsymbol{x_i}))^2 - \lambda^\star\|\boldsymbol{\theta}\|_2 \,.$$

Μεγιστοποιώντας την παραπάνω ποσότητα ως προς $\boldsymbol{\theta}$ παίρνουμε την εκτιμήτρια της κορυφής της ύστερης κατανομής ή MAP(Maximum a Posteriori), δηλαδή την $\hat{\boldsymbol{\theta}}$ που μεγιστοποιεί την σ.π.π. της ύστερης κατανομής. Αυτό το πρόβλημα όμως είναι ανάλογο της ποινικοποίησης που χρησιμοποιούσαμε στην συνάρτηση κόστους της Κλασσικής Στατιστικής για την Ευκλείδεια Νόρμα. Άρα χρησιμοποιώντας για πρότερη κατανομή την πολυδιάστατη τυποποιημένη κανονική είναι σαν να προσθέτουμε έναν όρο ποινικοποίησης στο μοντέλο ώστε να ελέγξουμε την προσαρμογή του. Ομοίως το ίδιο ισχύει και στην περίπτωση της κατηγοριοποίησης. Στο μοντέλο του PyMC3 χρησιμοποιήσαμε διάφορες τιμές για την παράμετρο $\lambda^\star = (0.5, 0.7, 1, 1.7, 2, 10, 100, 100^2)$ και καταλήξαμε ότι για $\lambda^\star = 1$ είχαμε καλύτερα αποτελέσματα ως προς την πρόβλεψη. Το μόνο που αλλάζει στον κώδικα, είναι οι γραμμές όπου δηλώνουμε τις πρότερες κατανομές για τα βάρη $\boldsymbol{\gamma}$ του πρώτου επιπέδου και τα βάρη $\boldsymbol{\beta}$ του κρυφού επιπέδου. Τώρα θα χρησιμοποιήσουμε την συνάρτηση $pm.Normal()$ με μηδενική μέση τιμή και μοναδιαία διασπορά, αντί της $pm.Flat()$.

```
# Weights from input to hidden layer
weights_in_1 = pm.Normal('w_in_1',0,sd=1,shape=(X_train.shape[1], n_hidden),
                                                    testval=init_1)
# Constants from input to hidden layer
constant_in_1 = pm.Normal('c_in_1',0,sd=1,shape=(n_hidden),testval=init_gamma)
# Weights from hidden layer to output
weights_1_out = pm.Normal('w_1_out',0,sd=1,shape=(n_hidden),testval=init_out)
# Constants from hidden layer to output
constant_in_2 = pm.Normal('c_1_out',0,sd=1,shape=(),testval=init_beta)
```

Κώδικας 2.7: Παραπάνω βλέπουμε τις αλλαγές στις μεταβλητές weights_in_1,constants_ in_1 και weights_1_out,constants_1_out που πρέπει να κάνουμε για να χρησιμοποιήσουμε ως πρότερη κατανομή την πολυδιάστατη Κανονική

Ο τρόπος επιλογής της παραμέτρου ποινικοποίησης $\lambda^\star$ που χρησιμοποιήσαμε για αυτό το μοντέλο γίνεται συνήθως με cross validation, αλλά στην περίπτωσή μας η πολυπλοκότητα του μοντέλου δεν βοηθούσε στο να γίνει αυτό εφικτό. Χρησιμοποιώντας την Μπεϋζιανή Στατιστική θα μπορούσαμε θεωρήσουμε το $\lambda^\star$ ως τυχαία μεταβλητή και να βρούμε την κατανομή του. Αυτά τα μοντέλα καλούνται Ιεραρχικά με την έννοια ότι τηρείται κάποια ιεραρχία, αφού πρέπει πρώτα να έχουμε μία τιμή π.χ. το $\lambda^\star$, την οποία θα χρησιμοποιήσουμε ως διασπορά της πρότερης κατανομής $N(0, \lambda^\star)$. Σε αυτήν την περίπτωση το $\lambda^\star$ καλείται υπερπαράμετρος του μοντέλου. Η επόμενη πρότερη κατανομή που θα αναφερθούμε

είναι ένα Ιεραρχικό μοντέλο και καλείται Automatic Relevance Determination Prior ή ARD. Οι Neal και Mackay (1995) προτείνουν αυτήν την πρότερη κατανομή για Μπεϋζιανά μοντέλα. Όπως είναι γνωστό αν $\boldsymbol{X} = (X_1 \ldots, X_m)$ είναι η επεξηγηματική μεταβλητή του μοντέλου τότε υπάρχει περίπτωση κάποια από τα χαρακτηριστικά $X_i$ να μην είναι σημαντικά και να μην παίζουν κανένα ρόλο στο μοντέλο. Το ARD μπορεί να αποφασίσει ποια από αυτά τα χαρακτηριστικά είναι σημαντικά ή όχι, δίνοντας πρότερες κατανομές $N(0, \sigma_i)$, με $\sigma_i \sim InvGamma(\frac{1}{\sigma_\star}, shape = \ldots)$ και $\sigma_\star \sim InvGamma()$, στα βάρη που σχετίζονται με την επεξηγηματική μεταβλητή. Στα νευρωνικά δίκτυα βάρη $\gamma$ από το επίπεδο εισόδου προς το κρυφό επίπεδο, σχετίζονται με την $X$. Ουσιαστικά η παράμετρος $\sigma_i$ ελέγχει την διασπορά της πρότερης κατανομής και άρα αν είναι πολύ μικρή το βάρος $\gamma_{ji}$ θα είναι μηδέν που σημαίνει ότι η μεταβλητή $X_i$ δεν επηρεάζει το μοντέλο. Η πρώτη υπερπαράμετρος $\sigma_\star$ είναι αυτή που ελέγχει την κατανομή της δεύτερης υπερπαραμέτρου $\sigma_i$. Στο δικό μας πρόβλημα θα χρησιμοποιήσουμε ένα πιο απλό Ιεραρχικό μοντέλο, αφού έχουμε σύνθετα δεδομένα των οποίων τα χαρακτηριστικά τα δημιουργήσαμε εμείς. Οι πρότερες κατανομές που θα δώσουμε στις παραμέτρους του μοντέλου μας είναι:

$$\begin{aligned} \gamma &\sim N_{15}(0, sd_1 I_{15}) \\ sd_1 &\sim InvGamma(a = 1, b = 1) \\ \beta &\sim N_6(0, sd_2 I_6) \\ sd_2 &\sim InvGamma(a = 1, b = 1). \end{aligned}$$

```
ann_input = theano.shared(X_train)
ann_output = theano.shared(Y_train)
# Set the number of hidden neurons
n_hidden=5
with pm.Model() as neural_network_h:
    # Initialize random weights between each layer
    init_1 = np.random.randn(X_train.shape[1], n_hidden)
    init_gamma = np.random.randn(n_hidden)
    init_out = np.random.randn(n_hidden)
    init_beta =np.random.randn()
    # Weights from input to hidden layer
    sd1=pm.InverseGamma('sd1',alpha=1,beta=1)
    weights_in_1 = pm.Normal('w_in_1',0,sd=sd1,shape=(X_train.shape[1], n_hidden),
                                                        testval=init_1)
    # Constants from input to hidden layer
    constant_in_1 = pm.Normal('c_in_1',0,sd=sd1,shape=(n_hidden),testval=init_gamma)
    sd2=pm.InverseGamma('sd2',alpha=1,beta=1)
    # Weights from hidden layer to output
    weights_1_out = pm.Normal('w_1_out',0,sd=sd2,shape=(n_hidden),testval=init_out)
    # Constants from hidden layer to output
    constant_in_2 = pm.Normal('c_1_out',0,sd=sd2,shape=(),testval=init_beta)
    # Build neural-network using tanh activation function
    act_1 = pm.math.tanh(pm.math.dot(ann_input,weights_in_1)+constant_in_1)

    act_out = pm.math.sigmoid(pm.math.dot(act_1,weights_1_out)+constant_in_2)
    # Binary classification -> Bernoulli likelihood
    out = pm.Bernoulli('out',
                       act_out,
                       observed=ann_output)
```

Κώδικας 2.8: Το αντίστοιχο μοντέλο στο PyMC3, αφού προσθέσαμε τις υπερπαραμέτρους $sd_1$ και $sd_2$.

## 2.3 Ύστερες Κατανομές

Φτάσαμε λοιπόν στο σημείο όπου πρέπει να εκτιμήσουμε την ύστερη κατανομή των παραμέτρων μας:

$$Pr(\boldsymbol{\theta}|D) = \frac{Pr(D|\boldsymbol{\theta})Pr(\boldsymbol{\theta})}{Pr(D)}.$$

Η πιο εύκολη περίπτωση είναι όταν η πρότερη είναι συζυγής ως προς την πιθανοφάνεια, δηλαδή η πρότερη και η ύστερη ανήκουν στην ίδια οικογένεια κατανομών. Ένα κλασσικό παράδειγμα είναι όταν η πρότερη και η πιθανοφάνεια είναι Κανονικές κατανομές, με αποτέλεσμα όταν συνδυαστούν η ύστερη που θα προκύψει να είναι και αυτή Κανονική. Αν τα πράγματα είναι τόσο απλά τότε έχουμε τελειώσει, αφού υπολογίσαμε αναλυτικά την ύστερη (κλειστή μορφή). Στην περίπτωση των νευρωνικών δικτύων δεν υπάρχουν κάποιες γνωστές συζυγής κατανομές και άρα θα πρέπει να υπολογίσουμε την ύστερη σε κλειστή μορφή. Εδώ το βασικό μας πρόβλημα είναι ο υπολογισμός του παρονομαστή $Pr(D) = \int_{\boldsymbol{\Theta}} Pr(D|\boldsymbol{\theta})Pr(\boldsymbol{\theta})d\boldsymbol{\theta}$. Ο υπολογισμός της παραπάνω ποσότητας, ιδίως σε μοντέλα με πολλές παραμέτρους είναι αρκετά δύσκολος και πολλές φορές αδύνατος. Μία πολύ καλή ιδέα, θα ήταν να μπορούσαμε με ΄κάποιον τρόπο΄, να παράγουμε τιμές από την ύστερη (Παραγωγή Τυχαίων Αριθμών ή Monte Carlo (MC) ) και στην συνέχεια να εκτιμήσουμε διάφορες ποσότητες που μας ενδιαφέρουν χρησιμοποιώντας δειγματικές εκτιμήσεις. Για παράδειγμα, αν θέλουμε να εκτιμήσουμε την μέση τιμή της πιθανότητας επιτυχίας $Pr(Y = 1|D)$ στο πιλοτικό πρόβλημα που δημιουργήσαμε με δεδομένα $D$, μπορούμε να προσομοιώσουμε $K$ τιμές $\boldsymbol{\theta^{(1)}}, \ldots, \boldsymbol{\theta^{(K)}}$ από την $Pr(\boldsymbol{\theta}|D)$ και να την εκτιμήσουμε με MC:

$$\hat{Pr}(Y = 1|D) = \sum_{i=1}^{K} Pr(Y = 1|\boldsymbol{\theta^{(i)}}).$$

Έπειτα θα δούμε δύο αλγορίθμους που μπορούμε να χρησιμοποιήσουμε για να παραγάγουμε τιμές από την ύστερη κατανομή του $\boldsymbol{\theta}$. Ο πρώτος αλγόριθμος ανήκει στην κατηγορία του Markov Chain Monte Carlo (MCMC) και ονομάζεται NUTS. Το MCMC, όπως φαίνεται και από το όνομά του, στηρίζεται στην παραγωγή τυχαίων αριθμών (MC) και στις Μαρκοβιανές αλυσίδες. Ο δεύτερος αλγόριθμος που θα δούμε ονομάζεται ADVI και είναι προσεγγιστική μέθοδος. Οι μέθοδοι που ανήκουν σε αυτήν την κατηγορία, βασιζόμενοι σε κάποιο κριτήρια, προτείνουν κάποια κατανομή ως την καλύτερη προσέγγιση της πραγματικής ύστερης κατανομής.

### 2.3.1 MCMC

Η ιδέα πίσω από το MCMC κρύβεται στις Μαρκοβιανές αλυσίδες. Μία Μαρκοβιανή αλυσίδα χαρακτηρίζεται από ένα χώρο καταστάσεων $\mathbb{X}$, όπου ανήκουν οι τιμές της αλυσίδας, ένα σύνολο $T$ που συνήθως είναι χρόνος και κάποιες πιθανότητες μετάβασης $p^{(n)}(x, y) = \mathbb{P}[X_n = x, X_n + 1 = y]$ που δηλώνουν την πιθανότητα η αλυσίδα να μεταβεί από την κατάσταση $x$ στην $y$ την χρονική στιγμή $n$. Σκοπός μας είναι να δημιουργήσουμε μία Μαρκοβιανή αλυσίδα με χώρο καταστάσεων τον παραμετρικό χώρο $\boldsymbol{\Theta}$ και πιθανότητες μετάβασης $p(\boldsymbol{\theta}, \boldsymbol{\theta'})$ τ.ω. η οριακή κατανομή της αλυσίδας να συγκλίνει στην ύστερη κατανομή $p(\boldsymbol{\theta}|D)$ των παραμέτρων μας. Αν το καταφέρουμε αυτό, θα έχουμε ένα δείγμα τιμών από την ύστερη κατανομή και άρα θα μπορούμε να εκτιμήσουμε διάφορα χαρακτηριστικά της χρησιμοποιώντας MC εκτιμήσεις. Η οριακή κατανομή της αλυσίδας, αν υπάρχει, θα πρέπει να είναι αναλλοίωτη, δηλαδή

$$\pi(\boldsymbol{\theta'}) = \sum_{\boldsymbol{\theta}} \pi(\boldsymbol{\theta}) p(\boldsymbol{\theta}, \boldsymbol{\theta'}) \qquad \text{για κάθε} \quad \boldsymbol{\theta'} \qquad (1).$$

Έτσι μετά από το σημείο όπου η σύγκλιση έχει επιτευχθεί, η αλυσίδα μας θα παράγει τιμές που ανήκουν στην $\pi$. Ακόμα πρέπει η αλυσίδα να είναι μη υποβιβάσιμη, ώστε να μην εγκλωβίζεται σε κάποιο υποσύνολο του χώρου καταστάσεων και γνησίως επαναληπτική, δηλαδή η μέση τιμή του χρόνου που θα κάνει η αλυσίδα να ξεκινήσει από την κατάσταση $x$ και να επιστρέψει σε αυτήν να είναι πεπερασμένη :

$$\mathbb{E}[T_x^+ | X_0 = x] < \infty.$$

Μια αλυσίδα που έχει και τις δύο παραπάνω ιδιότητες έχει μοναδική αναλλοίωτη κατανομή. Ένας τρόπος να εντοπίσουμε μία αναλλοίωτη κατανομή της αλυσίδας, είναι να βρούμε τις $p(\boldsymbol{\theta}, \boldsymbol{\theta}^\star)$ έτσι ώστε να ικανοποιούνται οι συνθήκες ακριβούς ισορροπίας

$$\pi(\boldsymbol{\theta}^\star) p(\boldsymbol{\theta}^\star, \boldsymbol{\theta}) = \pi(\boldsymbol{\theta}) p(\boldsymbol{\theta}, \boldsymbol{\theta}^\star).$$

Είναι γνωστό ότι αν η κατανομή $\pi$ και οι πιθανότητες $p$ είναι σε ακριβή ισορροπία, τότε η $\pi$ είναι αναλλοίωτη κατανομή για τη Μαρκοβιανή αλυσίδα. Ο πρώτος αλγόριθμος MCMC εφαρμόστηκε σε ένα πρόβλημα Στατιστικής Φυσικής από τον Metropolis et al,1953. Στην συνέχεια βελτιώθηκε από τον Hastings(1970). Αργότερα ακολούθησαν και άλλοι αλγόριθμοι βασιζόμενοι στον Metropolis-Hastings όπως ο αλγόριθμος του Gibbs, Slice Sampling, Hamiltonian Monte Carlo ή Hybrid Monte Carlo (HMC), Metropolis-adjusted Langevin algorithm (MALA) κ.α.

Ο αλγόριθμος Metropolis-Hastings μεταβαίνει από μία κατάσταση $\boldsymbol{\theta}$ στην επόμενη $\boldsymbol{\theta}'$ σύμφωνα με τα παρακάτω.

1. Σύμφωνα με κάποιες πιθανότητες μετάβασης $q(\boldsymbol{\theta}, \boldsymbol{\theta}^\star)$ προτείνεται μία επόμενη κατάσταση $\boldsymbol{\theta}^\star$, την οποία δεχόμαστε με πιθανότητα $a(\boldsymbol{\theta}, \boldsymbol{\theta}^\star)$. Άρα οι πιθανότητες μετάβασης που έχουμε επιλέξει είναι $p(\boldsymbol{\theta}, \boldsymbol{\theta}^\star) = q(\boldsymbol{\theta}, \boldsymbol{\theta}^\star)a(\boldsymbol{\theta}, \boldsymbol{\theta}^\star)$. Σκοπός μας είναι να βρούμε τις $p(\boldsymbol{\theta}, \boldsymbol{\theta}^\star)$ ώστε να ικανοποιείται η (1). Αν $\boldsymbol{\theta} = \boldsymbol{\theta}^\star$ τότε η (1) ισχύει. Αν $\boldsymbol{\theta} \neq \boldsymbol{\theta}^\star$, τότε χρησιμοποιώντας τις συνθήκες ακριβούς ισορροπίας έχουμε ότι

$$\begin{aligned}
&\pi(\boldsymbol{\theta}^\star)p(\boldsymbol{\theta}^\star, \boldsymbol{\theta}) = \pi(\boldsymbol{\theta})p(\boldsymbol{\theta}, \boldsymbol{\theta}^\star) \Rightarrow \\
&\pi(\boldsymbol{\theta}^\star)q(\boldsymbol{\theta}^\star, \boldsymbol{\theta})a(\boldsymbol{\theta}^\star, \boldsymbol{\theta}) = \pi(\boldsymbol{\theta})q(\boldsymbol{\theta}, \boldsymbol{\theta}^\star)a(\boldsymbol{\theta}, \boldsymbol{\theta}^\star) \Rightarrow \\
&\frac{a(\boldsymbol{\theta}, \boldsymbol{\theta}^\star)}{a(\boldsymbol{\theta}^\star, \boldsymbol{\theta})} = \frac{\pi(\boldsymbol{\theta}^\star)q(\boldsymbol{\theta}^\star, \boldsymbol{\theta})}{\pi(\boldsymbol{\theta})q(\boldsymbol{\theta}, \boldsymbol{\theta}^\star)}.
\end{aligned}$$

   Το οποίο μας δίνει την πιθανότητα αποδοχής:

$$a(\boldsymbol{\theta}, \boldsymbol{\theta}^\star) = min\left[1, \frac{\pi(\boldsymbol{\theta}^\star)q(\boldsymbol{\theta}^\star, \theta)}{\pi(\boldsymbol{\theta})q(\boldsymbol{\theta}, \boldsymbol{\theta}^\star)}\right]$$

2. Αν η $\boldsymbol{\theta}^\star$ γίνει αποδεκτή, τότε $\boldsymbol{\theta}' = \boldsymbol{\theta}^\star$, διαφορετικά $\boldsymbol{\theta}' = \boldsymbol{\theta}$.

Το προτέρημα αυτού του αλγορίθμου είναι ότι δεν χρειάζεται η σταθερά κανονικοποίησης $Pr(D)$, αφού εξαφανίζεται από το λόγο $\frac{\pi(\boldsymbol{\theta}^\star)}{\pi(\boldsymbol{\theta})}$ στην πιθανότητα αποδοχής. Στο δικό μας πρόβλημα, όπου έχουμε αρκετές παραμέτρους, ο M-H είχε πάρα πολύ κακή απόδοση. Για αυτό το λόγο επιλέξαμε τον πιο σύνθετο αλγόριθμο No-U-Turn Sampler(NUTS). Ο NUTS βασίζεται σε έναν άλλον MCMC αλγόριθμο, τον Hamiltonian Monte Carlo (HMC), ο οποίος χρησιμοποιεί τις εξισώσεις κίνησης Hamiltonian Dynamics για να προτείνει μία επόμενη τιμή $\boldsymbol{\theta}^\star$. Η λογική είναι η εξής: Μπορούμε να δούμε την παράμετρο του μοντέλου μας $\boldsymbol{\theta} \in \mathbb{R}^d$ , ως την θέση ενός σωματιδίου που κινείται στο χώρο με κάποια ταχύτητα $\boldsymbol{r} \in \mathbb{R}^d$. Έτσι μπορούμε να ορίσουμε την ολική συνάρτηση της ενέργειας $H$ ως το άθροισμα της δυναμικής ενέργειας $U(\boldsymbol{\theta})$ και της κινητικής ενέργειας $K(\boldsymbol{r})$ τ.ω. $H(\boldsymbol{\theta}, \boldsymbol{r}) = U(\boldsymbol{\theta}) + K(\boldsymbol{r})$. Η δυναμική ενέργεια του σωματιδίου μπορεί να ερμηνευτεί ως η αρνητική πιθανοφάνεια σε λογαριθμική κλίμακα της παραμέτρου $\boldsymbol{\theta}$, δηλαδή $U(\boldsymbol{\theta}) = -log(Pr(\boldsymbol{\theta}|D))$, ενώ η κινητική ενέργεια ως κάποια αυθαίρετη σ.π.π. Η εξισώσεις που διέπουν την κίνηση του σωματιδίου με $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_d)$ και $\boldsymbol{r} = (r_1, \ldots, r_d)$ σύμφωνα με την Χαμιλτονιανή δυναμική είναι:

$$\frac{\partial \theta_i}{\partial t} = \frac{\partial H}{\partial r_i} \quad \text{και} \quad \frac{\partial r_i}{\partial t} = \frac{\partial H}{\partial \theta_i}, \text{ όπου } i = 1, \ldots, d.$$

Όταν η αλυσίδα ανανεώνεται σύμφωνα με αυτές τις εξισώσεις, η οριακή κατανομή της είναι αναλλοίωτη. Για να δημιουργήσουμε μία τροχιά σύμφωνα με την Χαμιλτονιανές εξισώσεις στον υπολογιστή, θα πρέπει να χρησιμοποιήσουμε μία διακριτοποίηση που ονομάζεται LeapFrog Integrator. Χρησιμοποιώντας αυτήν την διακριτοποίηση, ουσιαστικά αφήνουμε το σωματίδιο με αρχική θέση $\boldsymbol{\theta} = \boldsymbol{\theta}_0$, να κάνει βήματα $\epsilon, 2\epsilon, \ldots, L\epsilon$ μακρυά από την αρχική του θέση. Ο αριθμός $L$ δηλώνει τον συνολικό αριθμό των βημάτων που θα κάνουμε και το $\epsilon$ το μήκος του κάθε βήματος. Για δοσμένο $L$ και $\epsilon$, η συνάρτηση $Leapfrog(\epsilon, L, \boldsymbol{\theta}_0, \boldsymbol{r}_0)$ είναι:

- Για $t = \epsilon, \ldots, L\epsilon$ και $i = 1, \ldots, d$
  1. $r_{t+\frac{\epsilon}{2},i} = r_{t,i} - (\epsilon/2)\frac{\partial U}{\partial \theta_i}(\theta_{t,i})$
  2. $\theta_{t+\epsilon,i} = \theta_{t,i} + \epsilon \frac{\partial K}{\partial r_i} r_{t+\frac{\epsilon}{2},i})$
  3. $r_{t+\epsilon,i} = r_{t+\frac{\epsilon}{2},i} - (\epsilon/2)\frac{\partial U}{\partial \theta_i}(\theta_{t+\epsilon,i})$
- Επέστρεψε $(\boldsymbol{\theta}_{L\epsilon}, \boldsymbol{r}_{L\epsilon})$

Η διαδικασία που περιγράψαμε παραπάνω, μας επιτρέπει να δημιουργήσουμε μία τροχιά $(\boldsymbol{\theta} = \boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_t, \ldots, \boldsymbol{\theta}^\star = \boldsymbol{\theta}_{L\epsilon})$ με τελική ταχύτητα $\boldsymbol{r}^\star$, που εξαρτάται από τις Χαμηλτονιανές εξισώσεις και μας δίνει την υποψήφια τιμή $\boldsymbol{\theta}^\star$. Για λόγους ευκολίας υποθέτουμε ότι η κινητική ενέργεια είναι $K(\boldsymbol{r}) = \boldsymbol{\theta}^T M^{-1} \boldsymbol{\theta}$ και άρα $\boldsymbol{r}|\boldsymbol{\theta} \sim N(0, M)$, όπου συνήθως χρησιμοποιούμε ως $M$ τον μοναδιαίο πίνακα $I_d$. Λόγω της Χαμιλτονιανής δυναμικής, η συνολική ενέργεια $H$ δεν διατηρείται με αποτέλεσμα να δεχόμαστε την τιμή $\boldsymbol{\theta}^\star$ με πιθανότητα αποδοχής:

$$a(\boldsymbol{\theta}, \boldsymbol{\theta}^\star) = min\Big[1, exp(-H(\boldsymbol{\theta}^\star, \boldsymbol{r}^\star) + H(\boldsymbol{\theta}, \boldsymbol{r}))\Big].$$

Χρησιμοποιώντας μία αρχική τιμή $\boldsymbol{\theta}^{(0)}$, τον αριθμό των τιμών $n$ που θέλουμε να προσομοιώσουμε από την ύστερη, το βήμα $\epsilon$, τον αριθμό των βημάτων $L$ και τον πίνακα $M \in \mathbb{R}^{d \times d}$, ο αλγόριθμος του HMC είναι ο εξής:

- Για $i = 1, \ldots, n$
  1. $\boldsymbol{r}^{(i)} \sim N(0, M)$
  2. $\boldsymbol{\theta}^{(i)} \leftarrow \boldsymbol{\theta}^{(i-1)}, \boldsymbol{\theta}^\star \leftarrow \boldsymbol{\theta}^{(i-1)}, \boldsymbol{r}^\star \leftarrow \boldsymbol{r}^{(i)}$
  3. Ανανεώνουμε το ζευγάρι $(\boldsymbol{\theta}^\star, \boldsymbol{r}^\star) = Leapfrog(\epsilon, L, \boldsymbol{\theta}^\star, \boldsymbol{r}^\star)$
  4. Θέτουμε $\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^\star$ με πιθανότητα $a(\boldsymbol{\theta}^{(i)}, \boldsymbol{\theta}^\star)$
- Επέστρεψε $\boldsymbol{\theta}^{(i)}$

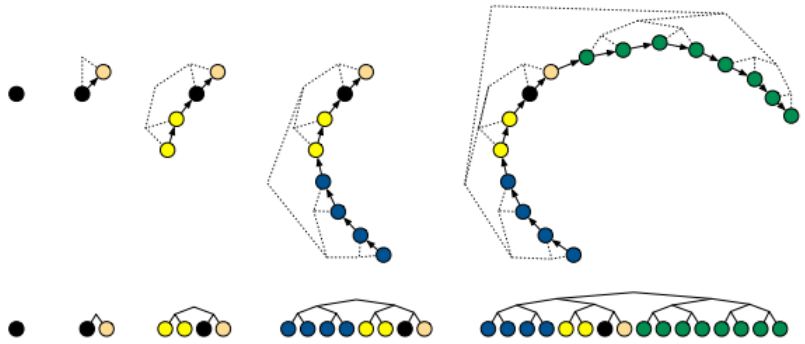
Ο HMC χρησιμοποιεί την πληροφορία από τις παραγώγους της ενέργειας $H$ για να κινείται μέσα στον χώρο με αποτέλεσμα να ξεφεύγει από την συμπεριφορά του τυχαίου περιπάτου που μπορεί να έχει ένα αλγόριθμος τύπου Metropolis-Hastings. Συνήθως έχει και αρκετά μεγάλη πιθανότητα αποδοχής. Γενικά η απόδοση του αλγορίθμου εξαρτάται από τις παραμέτρους $\epsilon$ και $L$ και αυτό είναι και το μεγάλο του μειονέκτημα. Ενώ για την παράμετρο $\epsilon$ υπάρχουν κάποιες λύσεις όπως Adaptive MCMC, η παράμετρος $L$ είναι το μεγαλύτερο πρόβλημα. Συγκεκριμένα όταν το $L$ είναι πολύ μικρό, ο αλγόριθμος έχει μία συμπεριφορά τυχαίου περιπάτου και η αλυσίδα δεν αναμιγνύεται καλά. Όταν το $L$ είναι αρκετά μεγάλο τότε οι τροχιές τείνουν να φτάσουν στο σημείο από το οποίο ξεκίνησαν, δηλαδή έχουν μία συμπεριφορά αναστροφής (U-turn). Ο αλγόριθμος NUTS αντιμετωπίζει αυτό το πρόβλημα της παραμέτρου $L$, ορίζοντας ένα κριτήριο $C$. Για παράδειγμα έστω ότι θέλουμε να μεταβούμε από το $\boldsymbol{\theta}$ στο $\boldsymbol{\theta}^\star$, τότε το κριτήριο είναι

$$C(\boldsymbol{\theta}, \boldsymbol{\theta}^\star) = \frac{\partial}{\partial t}(\boldsymbol{\theta}^\star - \boldsymbol{\theta})^T(\boldsymbol{\theta}^\star - \boldsymbol{\theta}) = (\boldsymbol{\theta}^\star - \boldsymbol{\theta})^T\frac{\partial}{\partial t}(\boldsymbol{\theta}^\star - \boldsymbol{\theta}) = (\boldsymbol{\theta}^\star - \boldsymbol{\theta})^T\boldsymbol{r},$$

που μας δείχνει πόσα βήματα $L$ ήταν αρκετά ώστε να αποφύγουμε την τάση της αναστροφής. Το κριτήριο αυτό είναι ανάλογο του κέρδους που θα είχαμε όσο περισσότερο απομακρυνόμαστε από την αρχική θέση $\boldsymbol{\theta}$ και άρα όταν φτάσει κάτω από το 0 σταματάμε την τροχιά. Επειδή όμως αυτό δεν ήταν αρκετό ώστε να εξασφαλιστεί η σύγκλιση της αλυσίδας, ο Neal(2003) έδωσε μία λύση χρησιμοποιώντας την εξής τεχνική. Ξεκινάμε από μία αρχική θέση $\boldsymbol{\theta}$ με αρχική ταχύτητα $\boldsymbol{r}$, διαλέγουμε μία κατεύθυνση τυχαία ( πίσω ή μπροστά ($v \sim Uniform(\{-1, 1\})$) σε φανταστικό χρόνο και πηγαίνουμε ένα βήμα LeapFrog($\epsilon$) αντίστοιχα. Στην συνέχεια κάνουμε το ίδιο για δύο βήματα LeapFrog( $\epsilon$), για τέσσερα κ.ο.κ, ώστε σύνολο να κάνουμε $2^j$ βήματα κάθε φορά, όπου $j$ ο αριθμός των επαναλήψεων διπλασιασμού μέχρι να φτάσουμε στο σημείο όπου η τροχιά του σωματιδίου ξεκινάει να αναστρέφεται, δηλαδή να ικανοποιείται η συνθήκη $C(\boldsymbol{\theta}, \boldsymbol{\theta}^\star) < 0$. Αυτή η διαδικασία του φανταστικού διπλασιασμού (Σχήμα 2.8) δημιουργεί ένα σύνολο $B$ με δυάδες θέσης-ταχύτητας ως στοιχεία του, τα οποία μπορούν να απεικονιστούν από Ισορροπημένα Δένδρα (Balanced Tree Graph). Με αυτό τον τρόπο και με τη βοήθεια της μεταβλητής $u|\boldsymbol{\theta}, \boldsymbol{r} \sim Uniform(0, exp(H(\boldsymbol{\theta}, \boldsymbol{r}))$ δημιουργούμε ένα σύνολο $C \subset B$, από τα ζευγάρια εκείνα που η από κοινού πιθανότητα $Pr(\boldsymbol{\theta}, \boldsymbol{r}|u)$ είναι ίδια. Αν δηλαδή $u \leq exp(H(\boldsymbol{\theta}', \boldsymbol{r}'))$, τότε όλη η τροχιά μέχρι και το $(\boldsymbol{\theta}', \boldsymbol{r}')$ ανήκει στην $C$. Στην συνέχεια διαλέγουμε ένα από αυτά τα ζευγάρια ως την επόμενη τιμή με πιθανότητα $\frac{1}{|C|}$. Χρησιμοποιώντας την διαδικασία, αυτή εξασφαλίζουμε ότι η αλυσίδα μας είναι χρονικά αντιστρέψιμη, δηλαδή ότι η από κοινού σ.π.π. της $(X_1, \ldots, X_n)$ είναι ίδια με της $(X_n, \ldots, X_1)$ για κάθε $n \geq 1$.

$$\mathbb{P}\{X_0 = x_0, \ldots, X_n = x_n\} = \mathbb{P}\{X_0 = x_n, \ldots, X_n = x_0\}.$$

Αν η αλυσίδα μας είναι χρονικά αντιστρέψιμη τότε ικανοποιούνται οι συνθήκες ακριβούς ισορροπίας (οι δύο όροι είναι ισοδύναμοι).



Σχήμα 2.9: Τα πάνω γραφήματα μας δίνουν τις θέσεις του σωματιδίου στον $\mathbb{R}^2$, όπως εξελίσσονται από τα αριστερά προς τα δεξιά για 1,2,3,4 βήματα LeapFrog. Οι κατεύθυνση του χρόνου ήταν $u = (1, -1, -1, 1)$. Στα κάτω γραφήματα φαίνονται τα Ισορροπημένα Δένδρα. Με αυτόν τον τρόπο μπορούμε να εντοπίσουμε την αναστροφή και στην συνέχεια να ξεχωρίσουμε τις θέσεις εκείνες που δεν παραβιάζουν την χρονική αντιστρεψιμότητα

### 2.3.2 Προσεγγιστικές Μέθοδοι

Σε πολύπλοκα μοντέλα, κυρίως όταν ανεβαίνει ο αριθμός των παραμέτρων, το MCMC χρειάζεται αρκετά μεγάλη υπολογιστική δύναμη για να προσομοιώσει τιμές από την ύστερη κατανομή. Αρκετές φορές η αλυσίδα μπορεί να έχει διάφορα προβλήματα και η σύγκλιση να μην έχει επιτευχθεί ή να αργεί πάρα πολύ. Σε αυτές τις περιπτώσεις μπορούμε να χρησιμοποιήσουμε κάποια προσεγγιστική μέθοδο, η οποία είναι σίγουρα ταχύτερη, αλλά όχι τόσο ακριβής. Η πιο γνωστή προσεγγιστική μέθοδο είναι αυτή του Laplace, όπου χρησιμοποιώντας το ανάπτυγμα Taylor δεύτερης τάξης γύρω από την κορυφή (MAP) της ύστερης κατανομής, παίρνουμε μία Κανονική Κατανομή ως προσέγγισή της. Μία άλλη κατηγορία προσεγγιστικών μεθόδων που χρησιμοποιείται αρκετά σε προβλήματα όπου ο αριθμός των παραμέτρων είναι αρκετά μεγάλος, είναι η Variational Approximation, γνωστή και ως VI(Variational Inference). Στην μέθοδο VI, διαλέγουμε μέσα από μία οικογένεια κατανομών $q(\boldsymbol{\theta}; \boldsymbol{\phi})$ με παράμετρο $\boldsymbol{\phi} \in \boldsymbol{\Phi}$,

την κατανομή εκείνη που είναι η καλύτερη προσέγγιση της ύστερης $Pr(\boldsymbol{\theta}|D)$. Η καλύτερη προσέγγιση προσδιορίζεται από την απόσταση KL γνωστή ως Kullback–Leibler divergence:

$$KL(Pr(\boldsymbol{\theta}|D)||q(\boldsymbol{\theta};\boldsymbol{\phi})) = \int_{-\infty}^{\infty} Pr(\boldsymbol{\theta}|D) log \frac{Pr(\boldsymbol{\theta}|D)}{q(\boldsymbol{\theta};\boldsymbol{\phi})} d\boldsymbol{\theta}.$$

Όσο μικρότερη είναι η απόσταση $KL(Pr(\boldsymbol{\theta}|D)||q(\boldsymbol{\theta};\boldsymbol{\phi}))$, τόσο καλύτερη η προσέγγιση $q(\boldsymbol{\theta};\boldsymbol{\phi})$ που προτείναμε, Στην πράξη, αφού η $q$ θεωρείται γνωστή έχουμε να λύσουμε ένα πρόβλημα βελτιστοποίησης:

$$\boldsymbol{\phi}^{\star} = \underset{\boldsymbol{\phi}\in\boldsymbol{\Phi}}{arg\,min} KL(Pr(\boldsymbol{\theta}|D) \;||\; q(\boldsymbol{\theta};\boldsymbol{\phi})).$$

Όμως επειδή η απόσταση $KL$ δεν μπορεί να γραφτεί σε κλειστή μορφή, θα χρησιμοποιήσουμε μία προσέγγισή της, το λεγόμενο Evidence Lower Bound ή (ELBO) το οποίο χρησιμοποιεί την $Pr(D,\boldsymbol{\theta}) = Pr(D|\boldsymbol{\theta})Pr(\boldsymbol{\theta})$

$$ELBO(\boldsymbol{\phi}) = \mathbb{E}_{q(\boldsymbol{\theta})}[log(Pr(D,\boldsymbol{\theta})] - \mathbb{E}_{q(\boldsymbol{\theta})}[log(q(\boldsymbol{\theta};\boldsymbol{\phi})].$$

Αντί να ελαχιστοποιήσουμε την απόσταση $KL$ ως προς $\boldsymbol{\phi}$, μπορούμε ισοδύναμα να μεγιστοποιήσουμε το ELBO. Το πρόβλημα με αυτή τη μέθοδο, είναι ότι πρέπει κάθε φορά που θέλουμε να υπολογίσουμε την ύστερη κατανομή να δημιουργούμε από την αρχή το μοντέλο του $VI$: Αυτό απαιτεί την κατάλληλη επιλογή της οικογένειας $q(\boldsymbol{\theta};\boldsymbol{\phi})$, τον ορισμό του προβλήματος βελτιστοποίησης (δηλαδή να υπολογίσουμε το ELBO και τις παραγώγους του) και την λύση του με κάποια αριθμητική μέθοδο. Επειδή αυτή η διαδικασία είναι αρκετά πολύπλοκη και απαιτείται μεγάλη εμπειρία για την υλοποίησή της, ο Gelman et all,(2016) πρότεινε τον αλγόριθμο Automatic Differentiation Variational Inference (ADVI). Ο ADVI χρειάζεται απλά ένα πιθανοκρατικό μοντέλο για να δημιουργήσει την μέθοδο VI, αλλά αυτό το μοντέλο θα πρέπει να είναι διαφορίσιμο (δηλαδή η παράμετρος $\boldsymbol{\theta}$ να είναι συνεχής και να υπάρχει η παράγωγος του $log(Pr(X,\boldsymbol{\theta})$ ως προς $\boldsymbol{\theta}$ στο σύνολο

$$supp(Pr(\boldsymbol{\theta})) = \{\boldsymbol{\theta}|\boldsymbol{\theta}\in\boldsymbol{\Theta} \quad \text{και} \quad Pr(\boldsymbol{\theta}) > 0\} \subseteq \mathbb{R}^d.$$

Το παραπάνω σύνολο είναι αρκετά σημαντικό, αφού δηλώνει σε ποια σημεία του παραμετρικού χώρου η πρότερη κατανομή είναι θετική και άρα καθορίζει και το $supp(Pr(\boldsymbol{\theta}|D))$. Γενικά υπάρχουν πολλά διαφορίσιμα πιθανοκρατικά μοντέλα (π.χ. Γενικευμένα Γραμμικά Μοντέλα, Μοντέλα Μίξης κατανομών, Γραμμικά δυναμικά μοντέλα, Γκαουσιανές Διαδικασίες κ.α) και για καλή μας τύχη συμπεριλαμβάνονται και τα νευρωνικά δίκτυα για παλινδρόμηση/κατηγοριοποίηση. Η ιδέα του ADVI είναι ότι πρώτα μετασχηματίζουμε της παραμέτρους του μοντέλου μας από το $supp(Pr(\boldsymbol{\theta}))$ στον $\mathbb{R}^d$ και στην συνέχεια επιλέγουμε μία

Κανονική οικογένεια κατανομών ως την προτεινόμενη προσέγγιση στον $\mathbb{R}^d$. Αντιστρέφοντας αυτην την διαδικασία έχουμε μία μη-Κανονική κατανομή ως την προτεινόμενη προσέγγιση για την ύστερη. Συγκεκριμένα θα χρησιμοποιήσουμε μία 1-1 και διαφορίσιμη συνάρτηση $T$ τ.ω.

$$T : supp(Pr(\boldsymbol{\theta})) \to \mathbb{R}^d.$$

Θέτουμε $\zeta = T(\boldsymbol{\theta})$ και άρα η απο κοινού μετασχηματισμένη σ.π.π. $g(D, \boldsymbol{\zeta})$ είναι πλέον συνάρτηση του $\boldsymbol{\zeta}$. Χρησιμοποιώντας τον Ιακοβιανό Πίνακα του $T^{-1}$ έχουμε ότι:

$$g(D, \boldsymbol{\zeta}) = Pr(D, T^{-1}(\boldsymbol{\zeta}))\left|detJ_{T^{-1}}(\boldsymbol{\zeta})\right|.$$

Στην συνέχεια προτείνουμε την Mean Field Approximation Κανονική κατανομή $q(\boldsymbol{\zeta}; \boldsymbol{\phi}) = N_d(\boldsymbol{\mu}, \sigma^2 I_d)$ ή την Full Rank Approximation $q(\boldsymbol{\zeta}; \boldsymbol{\phi}) = N_d(\boldsymbol{\mu}, \Sigma)$ ως την προσέγγιση της ύστερης κατανομής των $\boldsymbol{\zeta}$. Κάνοντας την πρώτη επιλογή, υποθέτουμε ότι οι παράμετροί μας είναι ανεξάρτητες μεταξύ τους, ενώ χρησιμοποιώντας την δεύτερη προσπαθούμε (μέσω του μη διαγώνιου πίνακα $\Sigma$)να ΄πιάσουμε΄ όσο το δυνατών καλύτερα τις συσχετίσεις μεταξύ των παραμέτρων. Παρόλο που η πρώτη επιλογή είναι πολύ πιο γρήγορη, εμείς θα χρησιμοποιήσουμε την δεύτερη που είναι πιο ακριβής. Για να διασφαλίσουμε ότι ο $\Sigma$ είναι θετικά ημιορισμένος και συμμετρικός, θα χρησιμοποιήσουμε παραγοντοποίηση Cholesky και άρα η προτεινόμενη κατανομή γίνεται $q(\boldsymbol{\zeta}; \boldsymbol{\phi}) = N_d(\mu, \Sigma = LL^T)$, δηλαδή $\boldsymbol{\phi} = (\boldsymbol{\mu}, L)$. Τώρα θα πρέπει να βρούμε την συνάρτηση του ELBO στον $\mathbb{R}^d$ και τις παραγώγους της, ώστε να προχωρήσουμε στο πρόβλημα μεγιστοποίησης. Η συνάρτηση του ELBO είναι :

$$ELBO(\boldsymbol{\phi}) = \mathbb{E}_{q(\boldsymbol{\zeta};\boldsymbol{\phi})}\left[Pr(D, T^{-1}(\boldsymbol{\zeta}))\left|detJ_{T^{-1}}(\boldsymbol{\zeta})\right|\right] + \mathbb{E}_{q(\boldsymbol{\zeta};\boldsymbol{\phi})}[log(q(\boldsymbol{\zeta}; \boldsymbol{\phi})].$$

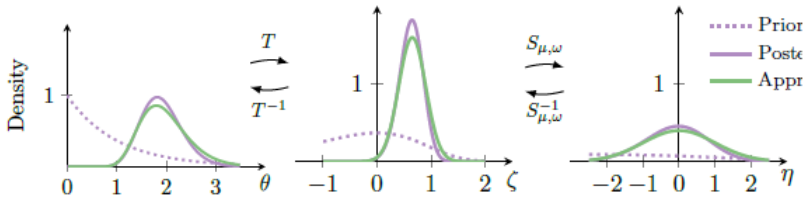
Για να αυτοματοποιήσουμε την διαδικασία και να μπορούμε να εισάγουμε τις παραγώγους της παραπάνω ποσότητας σε έναν υπολογιστή, θα χρησιμοποιήσουμε ακόμα ένα μετασχηματισμό $S_{\boldsymbol{\phi}}$ που ουσιαστικά μετασχηματίζει την Κανονική κατανομή σε τυποποιημένη Κανονική και μας διευκολύνει να υπολογίσουμε τις παραγώγους του ELBO. Στην περίπτωση του Full Rank έχουμε ότι $\boldsymbol{\eta} = S_{\boldsymbol{\phi}}(\boldsymbol{\zeta}) = L^{-1}(\boldsymbol{\zeta} - \boldsymbol{\mu})$ και άρα η κατανομή που προτείνεται τελικά είναι η

$$q(\boldsymbol{\eta}) = N_d(0, I_d).$$

Στο Σχήμα 2.9 βλέπουμε την διαδικασία των μετασχηματισμών που περιγράψαμε. Το τελικό πρόβλημα βελτιστοποίησης γίνεται

$$\boldsymbol{\phi}^\star = \underset{\boldsymbol{\phi}\in\boldsymbol{\Phi}}{arg\,max}\ \mathbb{E}_{q(\boldsymbol{\eta})}\left[Pr(D, T^{-1}(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}))\left|detJ_{T^{-1}}(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}))\right|\right] + \mathbb{E}_{q(\boldsymbol{\eta})}[log(q(\boldsymbol{\eta})].$$

Αφού χρησιμοποιήσουμε Monte Carlo για να εκτιμήσουμε τις παραγώγους $\nabla_\mu ELBO(\boldsymbol{\phi}), \nabla_L ELBO(\boldsymbol{\phi})$ μπορούμε να υπολογίσουμε τις παραμέτρους $\boldsymbol{\mu}$ και $L$ μέσω κάποιας αριθμητικής μεθόδου π.χ. Gradient Ascent ή Stochastic Gradient Ascent.




Σχήμα 2.10: Στην πρώτη εικόνα φαίνονται η πρότερη, η πραγματική ύστερη και η προσέγγισή της στον παραμετρικό χώρο Θ. Στην δεύτερη και στην τρίτη εικόνα βρίσκονται οι μετασχηματισμοί $T$ και $S_{\boldsymbol{\phi}}$ στον $\mathbb{R}^d$ αντίστοιχα. Στο συγκεκριμένα παράδειγμα έχουμε $d = 1$.

Ο αλγόριθμος του Full Rank ADVI για δεδομένα $D$, ένα μοντέλο $Pr(D, \boldsymbol{\theta})$ και $\rho$ την παράμετρο της Gradient Ascent έχει ως εξής:

1. $i = 1, \boldsymbol{\mu}^{(1)} = \mathbf{0}, L^{(1)} = I_d$
2. Όσο η διαφορά $\left|ELBO(\boldsymbol{\phi}^{(i+1)}) - ELBO(\boldsymbol{\phi}^{(i)})\right|$ είναι πάνω από κάποιο $\epsilon$:
   - Παράγουμε $M$ τιμές $\boldsymbol{\eta} \sim N_d(\mathbf{0}, I_d)$
   - Υπολογίζουμε με MC τις παραγώγους $\nabla_\mu ELBO(\boldsymbol{\phi})$ και $\nabla_L ELBO(\boldsymbol{\phi})$
   - Ανανεώνουμε τις παραμέτρους με Gradient Ascent
   - $\boldsymbol{\mu}^{(i+1)} \leftarrow \boldsymbol{\mu}^{(i)} + diag(\rho)\nabla_\mu ELBO$
   - $Ł^{(i+1)} \leftarrow Ł^{(i)} + diag(\rho)\nabla_L ELBO$
   - $i \leftarrow i + 1$
3. Επέστρεψε $\boldsymbol{\mu}^\star = \boldsymbol{\mu}^{(i)}$
4. Επέστρεψε $\Sigma^\star = L^{\star T} L^\star,\ L^\star = L^{(i)}$

Αντί της συνθήκης $\left|ELBO(\boldsymbol{\phi}^{(i+1)}) - ELBO(\boldsymbol{\phi}^{(i)})\right|$, μπορούμε να τρέξουμε τη μέθοδο για πάρα πολλές επαναλήψεις και ελέγξουμε τη σύγκλιση του $ELBO$ γραφικά. Μέχρις στιγμής έχουμε βρει τις παραμέτρους της κατανομής $q(\boldsymbol{\zeta}; (\boldsymbol{\mu}, \Sigma)$ $= N_d(\boldsymbol{\mu}, \Sigma)$. Στην συνέχεια μπορούμε να παράγουμε τιμές $\boldsymbol{\zeta}$ από αυτήν ( που

είναι πάρα πολύ γρήγορη διαδικασία γιατί είναι πλέον Κανονική) και χρησιμοποιώντας τον αντίστροφο του $T$ να τις μετασχηματίσουμε στο $\boldsymbol{\theta} = T^{-1}(\boldsymbol{\zeta})$, ώστε πλέον να έχουμε ένα δείγμα από την ύστερη κατανομή $Pr(\boldsymbol{\theta}|D)$.

### 2.3.3 Τα μοντέλα στο PyMC3

Τώρα θα εκτιμήσουμε την ύστερη κατανομή του $\boldsymbol{\theta}$ με τον αλγόριθμο Full Rank ADVI. Στο PyMC3 αυτό γίνεται εύκολα ανοίγοντας πρώτα το μοντέλο με την εντολή with. Στην συνέχεια εκχωρούμε σε μία μεταβλητή με το όνομα inferencefull την συνάρτηση που δηλώνει ποια μέθοδο (Full Rank ADVI ή Mean Field ADVI) θα διαλέξουμε. Τέλος προσαρμόζουμε το μοντέλο με την συνάρτηση pm.fit() με ορίσματα τον αριθμό των επαναλήψεων 200000 της SGD και τη μέθοδο της inferencefull, η οποία επιστρέφει τις παραμέτρους της κανονικής κατανομής στον τυποποιημένο πραγματικό χώρο.

```
%%time
with neural_network_flat:
    inferencefull= pm.FullRankADVI()
    approxfull= pm.fit(n=200000, method=inferencefull)
```

```
Average Loss = 26.111: 100%|██████████| 200000/200000 [03:37<00:00, 917.46it/s]
Finished [100%]: Average Loss = 26.123
```

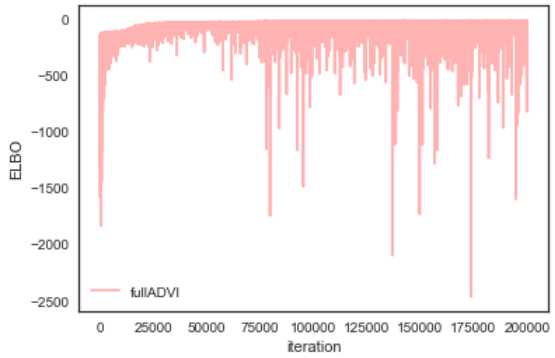
```
Wall time: 3min 45s
```

Κώδικας 2.11: Ο κώδικας για να βρούμε τις παραμέτρους $\boldsymbol{\phi}$ της προτεινόμενης κατανομής $q(\boldsymbol{\theta};\boldsymbol{\phi})$.

Για να δούμε πόσο χρόνο παίρνει στο μοντέλο να προσαρμοστεί θα χρησιμοποιούμε την εντολή $\%\%time$ η οποία τοποθετείται πριν από κάθε άλλη γραμμή κώδικα. Ο κώδικάς μας έτρεξε αρκετά γρήγορα σε μόλις 3 λεπτά και 45 δευτερόλεπτα. Τις τιμές του ELBO μπορούμε να τις πάρουμε μέσα από την προέκταση της inferencefull την inferencefull.hist. Εύκολα τώρα μπορούμε να δημιουργήσουμε ένα γράφημα με τις τιμές του ELBO ανά επανάληψη (Γράφημα 2.12). Από το Γράφημα 2.12 θα μπορούσαμε να συμπεράνουμε ότι ο αλγόριθμος έχει συγκλίνει. Βέβαια οι μεγάλες τιμές που βλέπουμε, προκύπτουν από την χρήση της Stochastic Gradient Descent. Για να πειστούμε ακόμα περισσότερο για τη σύγκλιση θα δείξουμε και τις τελευταίες 10000 επαναλήψεις (Γράφημα 2.13). Παρατηρούμε ότι η σύγκλιση έχει επιτευχθεί με μέση τιμή του ELBO στο 26.1. Στην συνέχεια θα παραγάγουμε 2000 τιμές από την ύστερη κατανομή του $\boldsymbol{\theta}$ χρησιμοποιώντας την approxfull.sample(). Αυτή η συνάρτηση κάνει δειγματοληψία από την κανονική κατανομή του τυποποιημένου πραγματικού χώρου με παραμέτρους αυτές που έχουν αποθηκευτεί στην μεταβλητή approxfull, στην

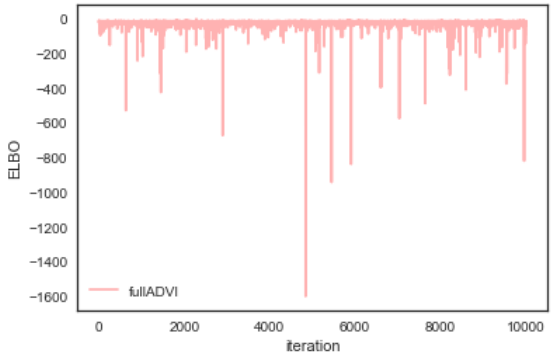
συνέχεια τις μετασχηματίζει ώστε να ανήκουν στον $\Theta$ και τις επιστρέφει σε ένα αντικείμενο τύπου trace.

```
plt.plot(-inferencefull.hist,'r',label='fullADVI', alpha=.3)
plt.legend()
plt.ylabel('ELBO')
plt.xlabel('iteration');
```




Γράφημα 2.12: Οι τιμές του ELBO ανά επανάληψη (0-200000).

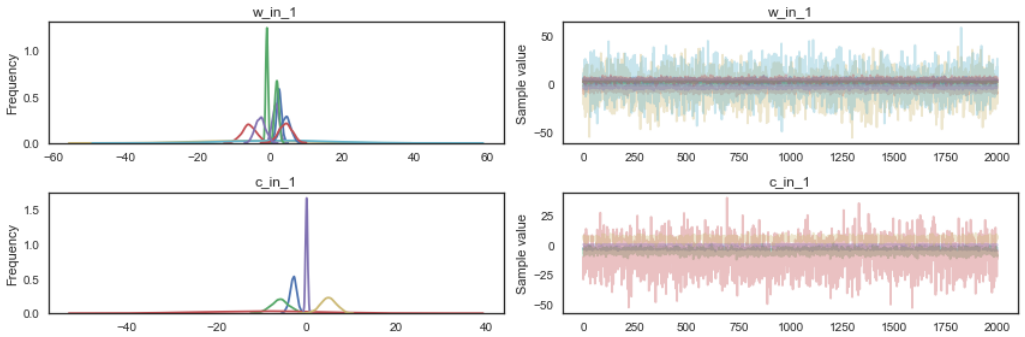
```
plt.plot(-inferencefull.hist[190000:],'r',label='fullADVI', alpha=.3)
plt.legend()
plt.ylabel('ELBO')
plt.xlabel('iteration');
```



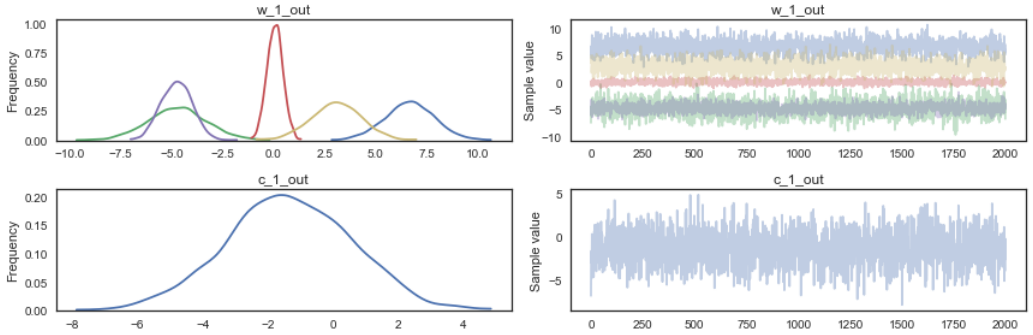

Γράφημα 2.13: Οι τιμές του ELBO ανά επανάληψη (190000-200000).

Χρησιμοποιώντας τη συνάρτηση pm.traceplot() μπορούμε να δούμε την κατανομή της $\boldsymbol{\theta}$ καθώς και ένα ιχνογράφημα (traceplot) των τιμών της. Στο Γράφημα 2.14 φαίνονται τα βάρη $\boldsymbol{\gamma}$ που είναι 15 στον αριθμό και στο Γράφημα 2.15 τα βάρη $\boldsymbol{\beta}$ τα οποία είναι 6. Τα ιχνογράμματα παρόλο που είναι συμπυκνωμένα φαίνονται αρκετά καλά για όλες τις παραμέτρους.

```
tracefull=approxfull.sample(2000)
pm.traceplot(tracefull);
```




Γράφημα 2.14: Οι ύστερες κατανομές των $\boldsymbol{\gamma}$, δηλαδή των παραμέτρων $\boldsymbol{\gamma_j}$ $\forall j = 1, \ldots, H = w\_in\_1$ και των σταθερές $\gamma_{j0}$ $\forall j = 1, \ldots, H = c\_in\_1$ και τα αντίστοιχα ιχνογράμματα.




Γράφημα 2.15: Οι ύστερες κατανομές των $\boldsymbol{\beta}$, δηλαδή των παραμέτρων $\beta_1, \ldots, \beta_H = w\_1\_out$ και της σταθερά $\beta_0 = c\_1\_out$ και τα αντίστοιχα ιχνογράμματα.

Οι ύστερες κατανομές για τα βάρη $\beta$ είναι εμφανέστατες και φαίνονται να είναι κανονικές με διαφορετική μέση τιμή η κάθε μία. Η σταθερά $\beta_0 = c\_1\_out$ φαίνεται να έχει πιο παχιές ουρές από τις άλλες, ενώ μπορούμε να δούμε ότι μία κατανομή (κόκκινο χρώμα) είναι κεντραρισμένη στο 0. Για τα βάρη $\gamma$ θα χρησιμοποιήσουμε την εντολή pm.summary(tracefull) για να δούμε και κάποια χαρακτηριστικά των κατανομών, αφού λόγω του μεγάλου αριθμού των παραμέτρων (15) οι κατανομές δεν φαίνονται και τόσο καλά. Όπως φαίνεται και από το γράφημα των σ.π.π των $\boldsymbol{\gamma}$ (Γράφημα 2.14), αλλά και από τα χαρακτηριστικά των ύστερων κατανομών (Σχήμα 2.16 και Σχήμα 2.17) οι κατανομές είναι συμμετρικές και οι περισσότερες τείνουν προς κανονικές με διαφορετικής κλίμακας

διασπορές, ενώ κάποιες άλλες είναι επίπεδες με μέση τιμή και διασπορά σχεδόν ίσες.

w_in_1:

| Mean | SD | MC Error | 95% HPD interval |
|---|---|---|---|
| | | [0, :] | |
| 2.579 | 0.697 | 0.015 | [1.242, 4.033] |
| -5.739 | 2.013 | 0.043 | [-9.909, -2.042] |
| -4.970 | 14.114 | 0.300 | [-30.193, 23.325] |
| 4.329 | 1.370 | 0.029 | [1.738, 7.039] |
| 4.319 | 1.850 | 0.041 | [0.710, 7.859] |
| | | [1, :] | |
| -0.813 | 0.317 | 0.007 | [-1.457, -0.203] |
| 1.627 | 0.911 | 0.019 | [-0.259, 3.290] |
| 2.434 | 14.169 | 0.280 | [-27.235, 28.823] |
| 1.910 | 0.573 | 0.012 | [0.798, 3.010] |
| -2.637 | 1.400 | 0.030 | [-5.384, -0.001] |

Posterior quantiles:

| 2.5 | 25 | 50 | 75 | 97.5 |
|---|---|---|---|---|
| | | [0, :] | | |
| 1.230 | 2.126 | 2.558 | 3.033 | 4.023 |
| -9.680 | -7.046 | -5.793 | -4.392 | -1.782 |
| -31.674 | -14.746 | -4.860 | 4.589 | 22.156 |
| 1.587 | 3.407 | 4.370 | 5.244 | 6.931 |
| 0.662 | 3.068 | 4.345 | 5.541 | 7.840 |
| | | [1, :] | | |
| -1.449 | -1.028 | -0.811 | -0.596 | -0.186 |
| -0.184 | 1.011 | 1.646 | 2.255 | 3.395 |
| -26.044 | -7.165 | 2.555 | 11.841 | 30.103 |
| 0.791 | 1.515 | 1.923 | 2.304 | 3.008 |
| -5.333 | -3.605 | -2.620 | -1.731 | 0.100 |

Σχήμα 2.16: Στην παραπάνω εικόνα βλέπουμε τον μέσο, την τυπική απόκλιση, το MC error και ένα 95% διάστημα αξιοπιστίας για τα βάρη $\boldsymbol{\gamma_j}$ $\forall j = 1, \ldots, H$ καθώς και τα $2.5, 25, 50, 75, 97.5$ ποσοστημόρια της ύστερης κατανομής τους.

c_in_1:

| Mean | SD | MC Error | 95% HPD interval |
|---|---|---|---|
| -2.936 | 0.770 | 0.016 | [-4.433, -1.323] |
| -5.920 | 2.000 | 0.044 | [-9.580, -1.792] |
| -10.152 | 12.796 | 0.319 | [-33.935, 14.739] |
| 0.003 | 0.234 | 0.005 | [-0.468, 0.435] |
| 4.875 | 1.730 | 0.037 | [1.455, 8.180] |

Posterior quantiles:

| 2.5 | 25 | 50 | 75 | 97.5 |
|---|---|---|---|---|
| -4.525 | -3.432 | -2.919 | -2.426 | -1.369 |
| -9.861 | -7.225 | -5.908 | -4.634 | -2.003 |
| -34.968 | -18.797 | -9.981 | -1.718 | 14.270 |
| -0.442 | -0.155 | 0.007 | 0.157 | 0.473 |
| 1.407 | 3.708 | 4.888 | 6.021 | 8.170 |

Σχήμα 2.17: Αντίστοιχα βλέπουμε τον μέσο, την τυπική απόκλιση, το MC error και ένα 95% διάστημα αξιοπιστίας για τις σταθερές $\gamma_{j0}$ $\forall j = 1, \ldots, H$ καθώς και τα $2.5, 25, 50, 75, 97.5$ ποσοστημόρια της ύστερης κατανομής τους.

Στην συνέχεια θα δούμε το μοντέλο με τις Κανονικές πρότερες κατανομές. Ακολουθώντας την ίδια διαδικασία με πριν, τρέχουμε πρώτα τον αλγόριθμο για το μοντέλο με την μοναδιαία διασπορά. Παρατηρούμε ότι η ελάχιστη τιμή του ELBO είναι μεγαλύτερη από το μοντέλο με τις flat (Κώδικας 2.18). Αυτό συμβαίνει λόγω των διαφορετικών πρότερων που χρησιμοποιήσαμε, κάνοντας το μοντέλο πιο πολύπλοκο ή επειδή το μοντέλο με τις flat, υπολογιστικά είναι αρκετά απλό.

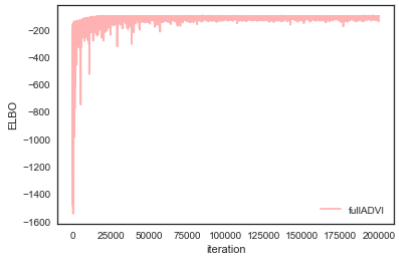
```
%%time
with neural_network:
    inferencefull= pm.FullRankADVI()
    approxfull= pm.fit(n=200000, method=inferencefull)
```

```
INFO (theano.gof.compilelock): Refreshing lock C:\Users\Nikos\AppData\Local\Theano\comp
ily_6_Model_69_Stepping_1_GenuineIntel-3.4.4-64\lock_dir\lock
Average Loss = 110.6: 100%|██████████| 200000/200000 [03:50<00:00, 869.53it/s]
Finished [100%]: Average Loss = 110.6
```

```
Wall time: 3min 56s
```

Κώδικας 2.18: Ο κώδικας για να βρούμε τις παραμέτρους $\boldsymbol{\phi}$ της προτεινόμενης κατανομής $q(\boldsymbol{\theta};\boldsymbol{\phi})$ για το μοντέλο με την πολυδιάστατη τυποποιημένη Κανονική πρότερη.

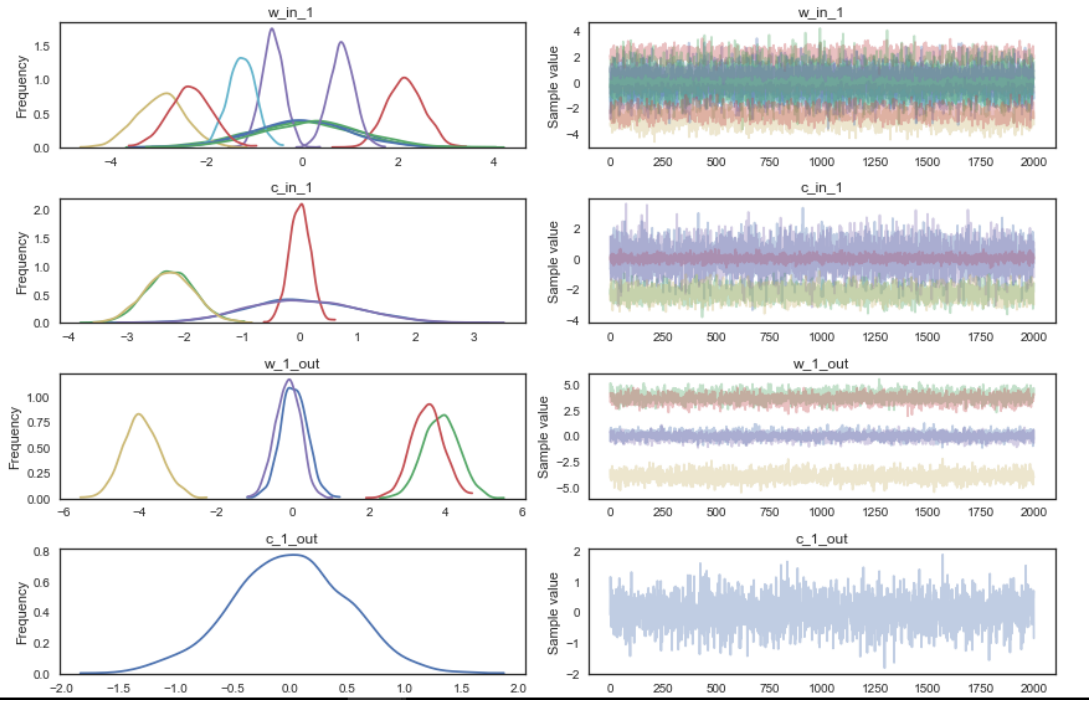
Βλέποντας το γράφημα με τις τιμές του ELBO (Γράφημα 2.19) για αυτό το μοντέλο θα μπορούσαμε να πούμε ότι η σύγκλιση έχει επιτευχθεί και μάλιστα με πιο ομαλό τρόπο από το μοντέλο με τις flat.




Γράφημα 2.19: Οι τιμές του ELBO ανά επανάληψη (0-200000) για το μοντέλο με την πολυδιάστατη τυποποιημένη Κανονική πρότερη.

Προσομοιώνουμε πάλι 2000 τιμές για κάθε παράμετρο. Η δειγματοληψία είναι αρκετά γρήγορη αφού απλά παράγουμε τιμές από κανονικές κατανομές και μετά τις μετασχηματίζουμε. Αυτό επηρεάζει και το ιχνόγραμμα το οποίο φαίνεται σχεδόν τέλειο (Γράφημα 2.20). Με σιγουριά θα λέγαμε ότι υπάρχει μεγάλη διαφορά σε σχέση με το πρώτο μοντέλο, πάλι όμως παρατηρούμε συμμετρικές κατανομές με διαφορετική ποικιλία ως προς το σχήμα τους. Το σχήμα των ύστερων των $\gamma$ είναι κάπως τριγωνικό, ενώ η παρουσία επίπεδων κατανομών είναι και εδώ αισθητή. Από την άλλη τα βάρη $\beta$ φαίνονται κανονικά κατανεμημένα αλλά με μικρότερη διασπορά. Η σταθερά $\beta_0$ έχει ίδιο σχήμα με αυτήν στο

προηγούμενο μοντέλο, άλλα με μηδενική μέση τιμή αυτή τη φορά. Σημαντικό είναι να αναφέρουμε ότι έχουμε αρκετές κατανομές με μέση τιμή 0. Η διαφορά εδώ από ένα κλασσικό γραμμικό μοντέλο είναι ότι αν η κατανομή κάποιας παραμέτρου $\theta_i$ για είναι κεντραρισμένη στο 0 αυτό δεν σημαίνει ότι η μεταβλητή που σχετίζεται με αυτήν είναι μη σημαντική. Στα νευρωνικά δίκτυα τα βάρη είναι αρκετά φορές πολύ κοντά στο 0, αφού σχετίζονται με την ενεργοποίηση πολλών νευρώνων και δεν είναι απλά συντελεστές ενός σταθμισμένου αθροίσματος.

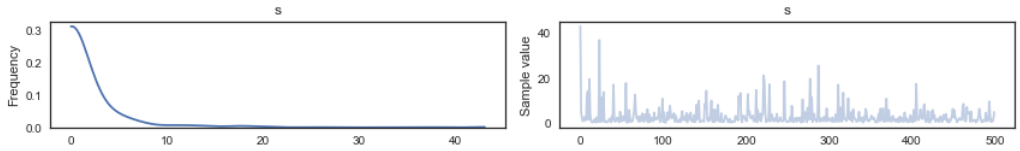



Γράφημα 2.20: Στην παραπάνω εικόνα βλέπουμε τις ύστερες κατανομές των $\boldsymbol{\gamma}$ καθώς και των $\boldsymbol{\beta}$ μαζί με τα αντίστοιχα ιχνογράμματα, για το μοντέλο με την τυποποιημένη Κανονική πρότερη.

Ομοίως τρέχουμε και το Ιεραρχικό μοντέλο με Κανονικές πρότερες με μέση τιμή $\mathbf{0}$ και διασπορά που ελέγχετε σε κάθε επίπεδο του δικτύου από τις μεταβλητές $sd\_1$ και $sd\_2$ που ακολουθούν την λίγο πληροφοριακή κατανομή Αντίστροφη Γάμμα με παραμέτρους $\alpha = \beta = 1$. Αυτή η κατανομή φαίνεται στο Γράφημα 2.21. Η παράμετρος του μοντέλου μας αυτή τη φορά είναι $\boldsymbol{\theta} = (\boldsymbol{\gamma}, \boldsymbol{\beta}, sd\_1, sd\_2) \in \mathbb{R}^{23}$. Αφού το μοντέλο έχει δύο παραπάνω παραμέτρους, αργεί λίγο παραπάνω από τα προηγούμενα. Όπως βλέπουμε από το Γράφημα 2.22 η σύγκλιση έχει επιτευχθεί. Η ελάχιστη τιμή του $ELBO$ σε αυτήν την περίπτωση είναι 86.3

```
with pm.Model() as G:
    s=pm.InverseGamma('s',alpha=1,beta=1)
    inference2= pm.FullRankADVI()
    approx2= pm.fit(n=20000, method=inference2)
t=approx2.sample(500)
pm.traceplot(t);
```

```
Average Loss = 0.14304: 100%|██████████| 20000/20000 [00:08<00:00, 2362.93it/s]
Finished [100%]: Average Loss = 0.14256
```



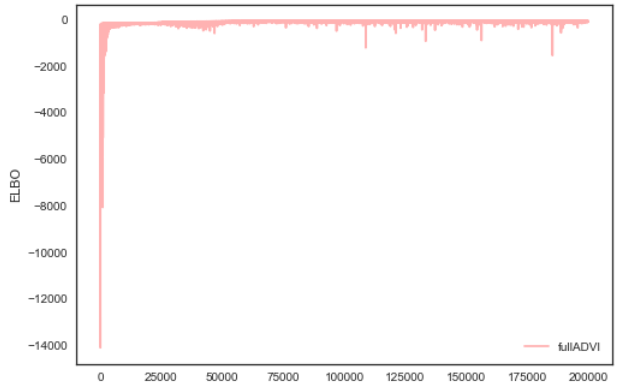

Γράφημα 2.21: Η Αντίστροφη Γάμμα κατανομή με παραμέτρους $\alpha = \beta = 1$

```
%%time
with neural_network_h:
    inferencefull= pm.FullRankADVI()
    approxfull= pm.fit(n=200000, method=inferencefull)
```

```
WARNING (theano.gof.compilelock): Overriding existing lock by dead process '12908' (I am p
Average Loss = 86.241: 100%|██████████| 200000/200000 [04:21<00:00, 763.58it/s]
Finished [100%]: Average Loss = 86.248
```

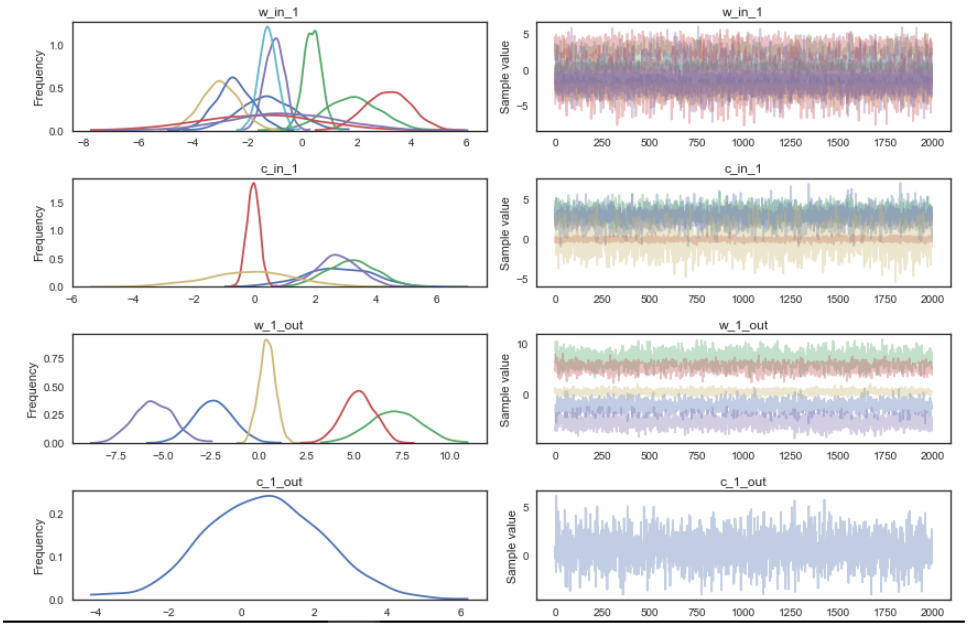
```
Wall time: 4min 42s
```

```
plt.plot(-inferencefull.hist,'r',label='fullADVI', alpha=.3)
plt.legend()
plt.ylabel('ELBO')
plt.xlabel('iteration');
```

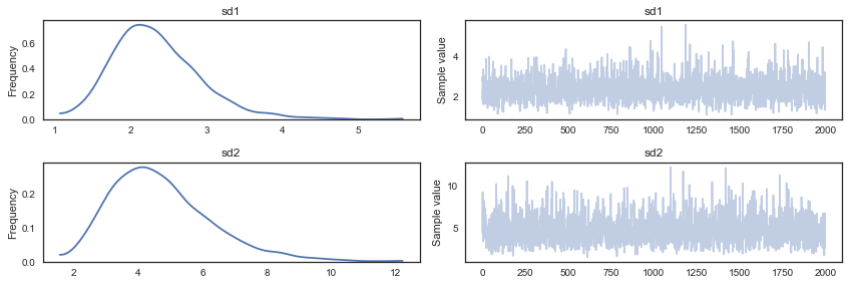



Γράφημα 2.22: Ο κώδικας για να βρούμε τις παραμέτρους $\boldsymbol{\phi}$ της προτεινόμενης κατανομής $q(\boldsymbol{\theta}; \boldsymbol{\phi})$ για το Ιεραρχικό μοντέλο, καθώς και οι τιμές του ELBO ανά επανάληψη.

Συνεχίζουμε με τις ύστερες κατανομές του $\boldsymbol{\theta}$. Βλέποντας το Γράφημα 2.23 θα μπορούσαμε να πούμε ότι κάθε ύστερη κατανομή έχει διαφορετική μορφή, σε αντίθεση με τα προηγούμενα μοντέλα όπου πολλές κατανομές ήταν ακριβώς ίδιες. Βέβαια αυτό προκύπτει επειδή αφήσαμε την διασπορά των πρότερων σε κάθε επίπεδο να παίζει και όχι να είναι αυστηρά ίση με τη μονάδα. Οι ύστερες κατανομές των υπερπαραμέτρων $sd\_1$ και $sd\_2$ φαίνονται στο Γράφημα 2.24.




Γράφημα 2.23: Στην παραπάνω εικόνα βλέπουμε τις ύστερες κατανομές των $\boldsymbol{\gamma}$ καθώς και των $\boldsymbol{\beta}$ μαζί με τα αντίστοιχα ιχνογράμματα, για το Ιεραρχικό μοντέλο.



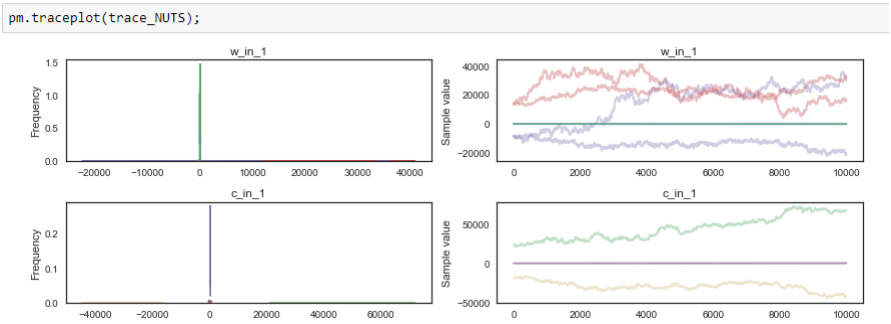

Γράφημα 2.24: Οι ύστερες κατανομές των υπερπαραμέτρων του Ιεραρχικού μοντέλου $sd1$ και $sd2$, καθώς και τα αντίστοιχα ιχνογράμματα. Η κατανομή της υπερπαραμέτρου των $\boldsymbol{\beta}$ φαίνεται να έχει την ίδια κατανομή με αυτήν των $\boldsymbol{\gamma}$, αλλά με διπλάσια μέση τιμή.

Αφού τελειώσαμε με τον ADVI, θα προσπαθήσουμε να υπολογίσουμε την ύστερη κατανομή χρησιμοποιώντας τον MCMC αλγόριθμο NUTS. Η διαδικασία στο PyMC3 είναι λίγο διαφορετική αυτή τη φορά. Παρόλο που το μοντέλο παραμένει ίδιο, η δειγματοληψία γίνεται κατευθείαν με χρήση της συνάρτησης pm.sample(), η οποία παίρνει σαν ορίσματα τον αριθμό των τιμών που θέλουμε να προσομοιώσουμε και τον αλγόριθμο MCMC π.χ. Metropolis, Gibbs, NUTS, HMC κ.α καθώς και μία αρχική τιμή. Στο PyMC3 ο προκαθορισμένος αλγόριθμος είναι ο NUTS και η αρχική τιμή καθορίζεται χρησιμοποιώντας τον ADVI για 200000 επαναλήψεις. Το τελευταίο βήμα είναι απαραίτητο, αφού ο NUTS έχει μεγάλο πρόβλημα αρχικοποίησης. Αντί του ADVI χρησιμοποιήσαμε και την εκτιμήτρια MAP για να βρούμε την αρχική τιμή του $\boldsymbol{\theta}$ και μετά να τρέξουμε τον NUTS, αλλά τα αποτελέσματα ήταν ίδια. Ξεκινάμε με το μοντέλο με τις flat πρότερες κατανομές, προσομοιώνοντας 10000 τιμές για κάθε παράμετρο στο μοντέλο μας. Ο αλγόριθμος χρησιμοποιεί ως burn in 500 επαναλήψεις και μετά ξεκινάει να κρατάει τιμές.

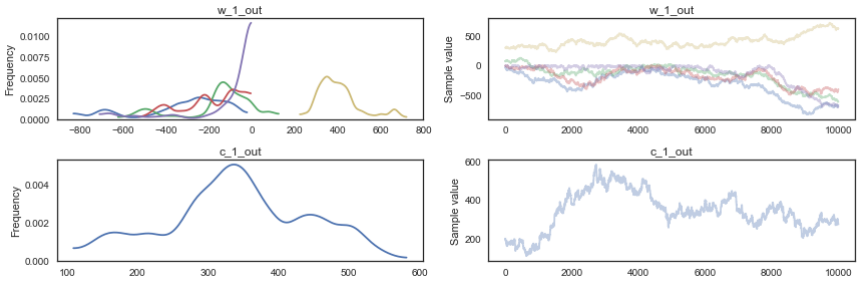
```
: with neural_network_flat:
      trace_NUTS=pm.sample(10000)

Auto-assigning NUTS sampler...
Initializing NUTS using ADVI...
Average Loss = 24.77:  90%|████████▉ | 180024/200000 [01:49<00:10, 1844.90it/s]
Convergence archived at 180100
Interrupted at 180,100 [90%]: Average Loss = 46.168
INFO (theano.gof.compilelock): Refreshing lock C:\Users\Nikos\AppData\Local\Theano\
ily_6_Model_69_Stepping_1_GenuineIntel-3.4.4-64\lock_dir\lock
100%|████████████████████████████████| 10500/10500 [1:10:58<00:00,  2.49it/s]
```

Κώδικας 2.25: Η δειγματοληψία με τον αλγόριθμο NUTS για το μοντέλο με την flat πρότερη κατανομή

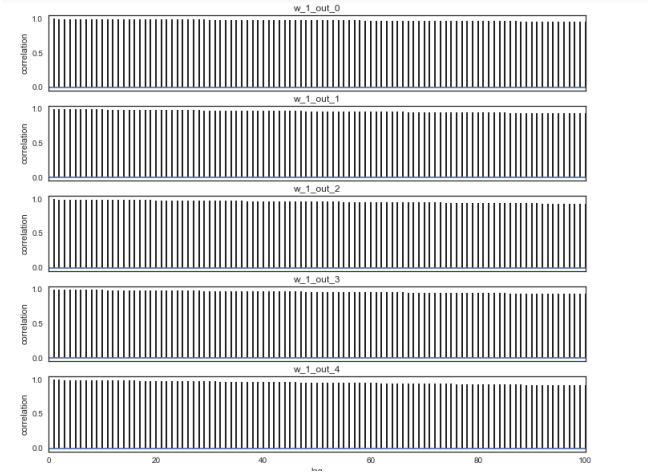
Συνολικά η διαδικασία της δειγματοληψίας πήρε πάνω από μία ώρα να τρέξει, κάτι το οποίο περιμέναμε λόγω του μεγάλου αριθμού των παραμέτρων. Αφού έχουμε τις τιμές μας, μπορούμε τώρα να εκτιμήσουμε διάφορα χαρακτηριστικά της ύστερης χρησιμοποιώντας δειγματικές εκτιμήσεις. Στο Γράφημα 2.26 και στο Γράφημα 2.27 φαίνονται τα γραφήματα των ύστερων κατανομών και το ιχνόγραμμα των τιμών που προσομοιώσαμε με τον NUTS. Παρατηρούμε ένα είδος συσχέτισης της κάθε τιμής $x_{t+1}$ με τις προηγούμενες $x_t, \ldots, x_{t-k}$. Αυτό μπορούμε να το δούμε και από το διάγραμμα αυτοσυσχέτισης των τιμών που προσομοιώσαμε στο PyMC3 με την εντολή pm.autocorrplot(). Στο Γράφημα 2.28 μπορούμε να δούμε την αυτοσυσχέτιση για τις παραμέτρους $\beta_1, \beta_2, \beta_3, \beta_4$ και $\beta_5$ .




Γράφημα 2.26: Οι ύστερες κατανομές για τα βάρη $\boldsymbol{\gamma}$ και τα αντίστοιχα ιχνογράμματα.




Γράφημα 2.27: Οι ύστερες κατανομές για τα βάρη $\boldsymbol{\beta}$ και τα αντίστοιχα ιχνογράμματα.




Γράφημα 2.28: Η αυτοσυσχέτιση είναι 1 για πάνω από 100 τιμές και ίσως παραπάνω για τα βάρη $\beta_1, \ldots, \beta_H$.

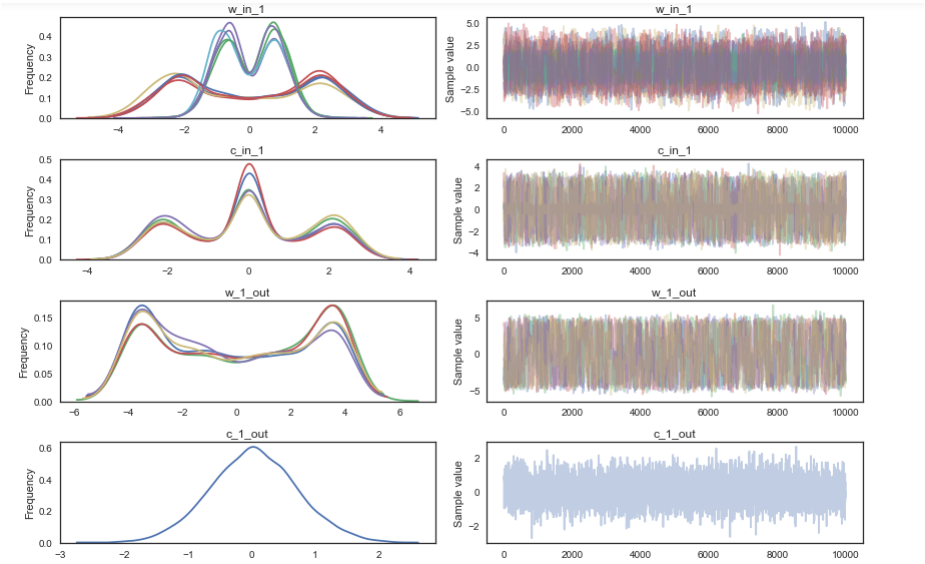
Συνήθως το πρόβλημα αυτοσυσχέτισης λύνεται αν δεχόμαστε τιμές ανά $k$, αλλά σε αυτήν την περίπτωση δεν θα αλλάξει κάτι. Ένας από τους λόγους όπου η αλυσίδα μας δεν έχει συγκλίνει μπορεί να είναι η επιλογή των flat πρότερων αφού πρέπει να τεθούν πρώτα κάποιοι περιορισμοί ώστε η ύστερη να είναι αποδεκτή ως κατανομή. Επίσης μπορεί να υπάρχει συσχέτιση μεταξύ των παραμέτρων, πράγμα που κάνει την δειγματοληψία στον πολυδιάστατο παραμετρικό χώρο αρκετά δύσκολη ακόμα και για έναν πολύ δυνατό αλγόριθμο. Τέλος θα παρουσιάσουμε και το μοντέλο με την τυποποιημένη πρότερη Κανονική κατανομή που τα αποτελέσματα ήταν σαφώς καλύτερα. Για χάρη συντομίας δεν θα παρουσιάσουμε το Ιεραρχικό μοντέλο. Τα αποτελέσματα του ήταν καλύτερα από το μοντέλο με τις flat όμως αυτή τη φορά η αλυσίδα φαινόταν να μην δεχόταν αρκετές τιμές για μεγάλα χρονικά διαστήματα. Έτσι λόγω τις μεγάλης πολυπλοκότητας, το μοντέλο έκανε πάνω από τρεις ώρες να τρέξει χρησιμοποιώντας σχεδόν το 100% της μνήμης του υπολογιστή. Ξεκινάμε λοιπόν με το μοντέλο neural network, προσομοιώνοντας πάλι 10000 τιμές (Κώδικας 2.29).

```
with neural_network:
    trace_NUTS2=pm.sample(10000)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using ADVI...
Average Loss = 183.82:   5%|▌         | 10242/200000 [00:06<01:49, 1733.10it/s]
Convergence archived at 10300
Interrupted at 10,300 [5%]: Average Loss = 245.36
INFO (theano.gof.compilelock): Refreshing lock C:\Users\Nikos\AppData\Local\Theano\
ily_6_Model_69_Stepping_1_GenuineIntel-3.4.4-64\lock_dir\lock
100%|██████████████████████████████████| 10500/10500 [13:58<00:00, 12.53it/s]
```

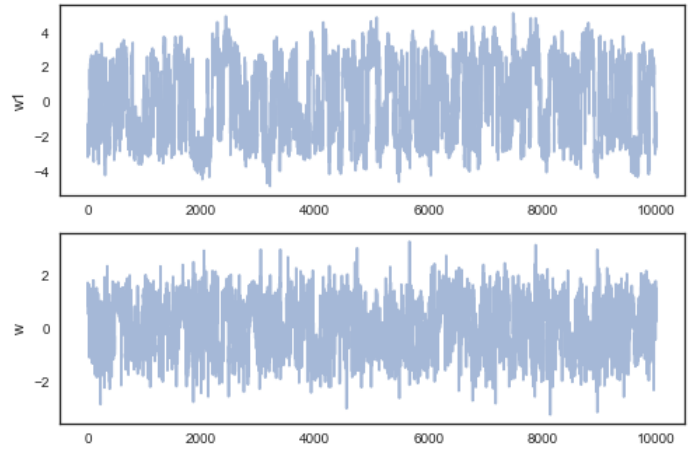
Κώδικας 2.29: Η δειγματοληψία με τον NUTS για το μοντέλο με πρότερη την τυποποιημένη Κανονική κατανομή. Από την αρχή βλέπουμε ότι το μοντέλο έτρεξε αρκετά γρήγορα σε μόλις 14 λεπτά ( και λίγο παραπάνω για την αρχικοποίηση).

Στο Γράφημα 2.30 φαίνονται οι ύστερες κατανομές των παραμέτρων μας και τα αντίστοιχα ιχνογράμματα των τιμών που προσομοιώσαμε με το MCMC. Για δεύτερη φορά η χρήση του MCMC μας εντυπωσιάζει. Οι ύστερες κατανομές φαίνονται να είναι όλες πολυκόρυφες και με εξαίρεση την μεταβλητή w_in_1, όλες έχουν το ίδιο σχήμα. Επειδή τα ιχνογράμματα δεν φαίνονται σχεδόν καθόλου θα επιλέξουμε τυχαία δύο παραμέτρους από το διάνυσμα $\theta$ και θα δημιουργήσουμε τα ιχνογραμματά τους. Θα επιλέξουμε δύο παραμέτρους από τα βάρη $\gamma$ που το γράφημά τους φαίνεται διαφορετικό. Τα αποτελέσματα φαίνονται στο Γράφημα 2.31.




Γράφημα 2.30: Οι ύστερες κατανομές για τα βάρη $\boldsymbol{\gamma}$ και τα βάρη $\boldsymbol{\beta}$ και τα αντίστοιχα ιχνογράμματα, όπως προσομοιώθηκαν με τον NUTS για το μοντέλο με πρότερη την τυποποιημένη Κανονική κατανομή.

```
fig, axs = plt.subplots(nrows=2)
axs[0].plot(trace_NUTS2.get_values('w_in_1')[:,0,0], alpha=.5);
axs[0].set(ylabel='w1');
axs[1].plot(trace_NUTS2.get_values('w_in_1')[:,1,2], alpha=.5);
axs[1].set(ylabel='w');
```




Γράφημα 2.31: Τα ιχνογράμματα δυο παραμέτρων από τα βάρη $\boldsymbol{\gamma}$, οι οποίες επιλέχθηκαν τυχαία. Παρατηρούμε ότι τα ιχνογράμματα είναι αρκετά καλά. Οι εναλλαγές των τιμών που βλέπουμε π.χ. από το 4 στο -4 στην πρώτη παράμετρο ή από το 2 στο -2 για την δεύτερη, ταυτίζονται με το ότι οι ύστερες κατανομές είναι πολυκόρυφες.

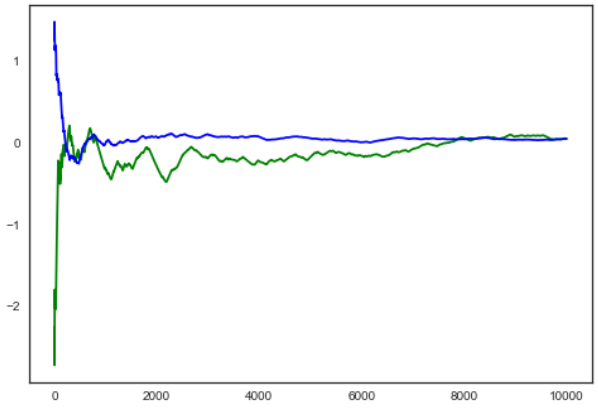
Για να βεβαιωθούμε ότι η σύγκλιση έχει επιτευχθεί θα δημιουργήσουμε μία συνάρτηση που να μας επιστρέφει τον εργοδικό μέσο της αλυσίδας της κάθε παραμέτρου $\boldsymbol{\theta}$.

```
def ergodic_mean(z):
    h=[]
    for i in range(1,len(z)+1):
        zz=np.cumsum(z)/i
        h.append(zz[i-1])
    return h
```

Κώδικας 2.32: Ο κώδικας για τον εργοδικό μέσο.

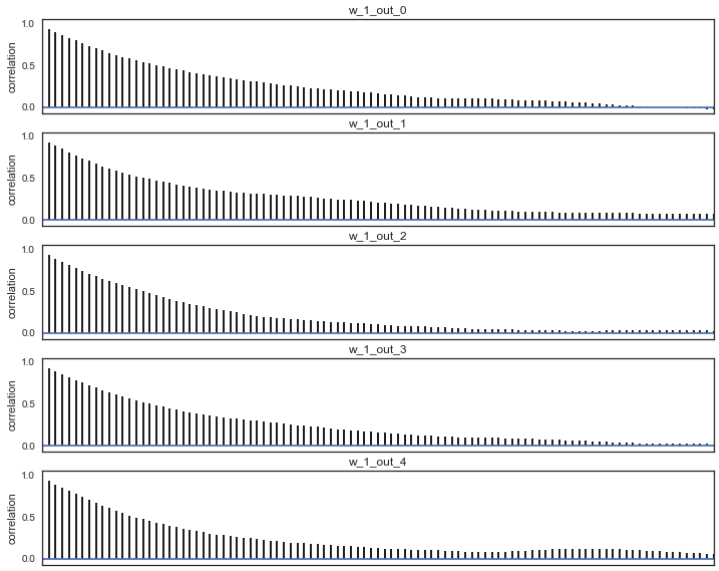
Τώρα θα δημιουργήσουμε το γράφημα του εργοδικού μέσου των δυο παραμέτρων από τα βάρη $\boldsymbol{\gamma}$ που χρησιμοποιήσαμε για τα δυο παραπάνω ιχνογράμματα.

```
R_M=ergodic_mean(trace_NUTS2.get_values(varname='w_in_1')[:,0,0])
R_M2=ergodic_mean(trace_NUTS2.get_values(varname='w_in_1')[:,1,2])
fig, ax1 = plt.subplots()
ax2 = ax1
ax1.plot(range(0,10000),R_M, 'g-');
ax2.plot(range(0,10000),R_M2, 'b-');
```




Γράφημα 2.33: Ο εργοδικός μέσος για τις δύο παραμέτρους από τα βάρη $\boldsymbol{\gamma}$. Όπως βλέπουμε και για τις δύο αλυσίδες ο εργοδικός μέσος φαίνεται να έχει συγκλίνει στην τιμή 0, όπως ήταν και το αναμενόμενο αφού και οι δύο ύστερες είναι συμμετρικές ως προς το κέντρο.

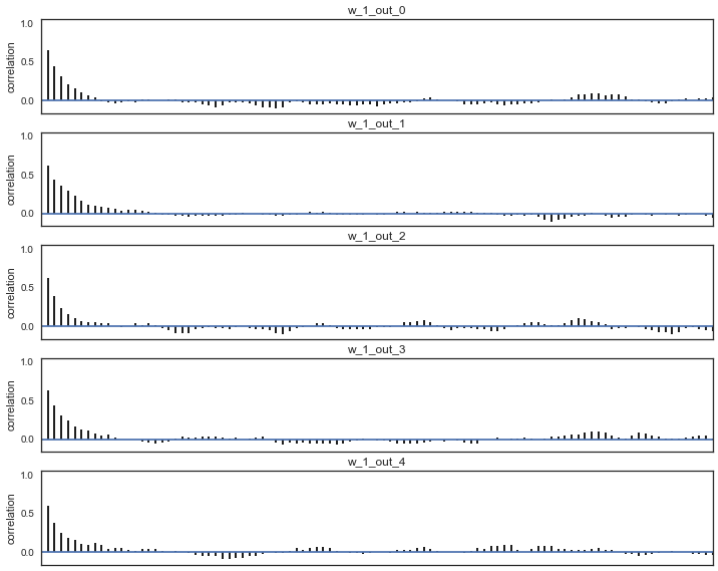
Θα συνεχίσουμε με το γράφημα αυτοσυσχέτισης των παραμέτρων. Ας πάρουμε για παράδειγμα τα βάρη $\boldsymbol{\beta}$ (χωρίς την σταθερά $\beta_0$), όπως ακριβώς κάναμε και για το μοντέλο με τις flat πρότερες (Γράφημα 2.34).



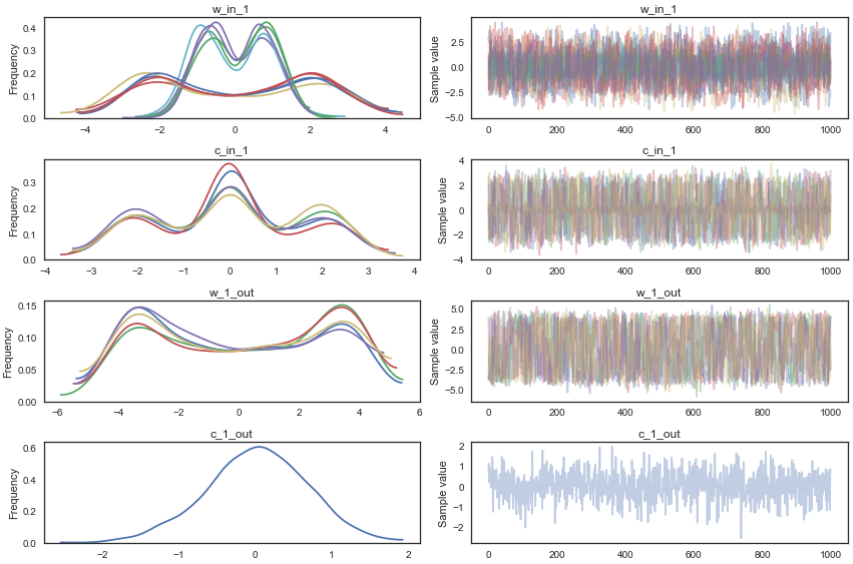

Γράφημα 2.34: Το διάγραμμα αυτοσυσχέτισης για τα βάρη $\beta_1, \ldots, \beta_H$.

Τα αποτελέσματα φαίνονται ικανοποιητικά. Για να μειώσουμε στο ελάχιστο την αυτοσυσχέτιση των τιμών που προσομοιώσαμε μπορούμε να πάρουμε τιμές ανά $k = 10$. Βέβαια αυτή η κίνηση θα αλλάξει την συσχέτιση που έχουνε οι τιμές μεταξύ τους και όχι το σχήμα ή οποιαδήποτε άλλο χαρακτηριστικό των ύστερων κατανομών. Για να πάρουμε τις τιμές ανά 10 μπορούμε να χρησιμοποιήσουμε τους δείκτες : μέσα στο πρώτο όρισμα της μεταβλητής trace_NUTS2. Η εντολή στο PyMC3 είναι $pm.autocorrplot(trace_NUTS2[:: 10], varnames = [\prime w_1_out\prime])$και τα αποτελέσματά της φαίνονται στο Γράφημα 2.35. Το πρόβλημα της αυτοσυσχέτισης όπως παρατηρούμε λύθηκε, όμως οι ύστερες κατανομές είναι ακριβώς οι ίδιες (Γράφημα 2.36). Βέβαια τώρα εκτιμάμε τις κατανομές μας μέσα από 1000 τιμές και όχι 10000 όπως στην αρχή.

Γιατί όμως οι ύστερες που παίρνουμε με τον NUTS είναι πολυκόρυφες, ενώ με τον ADVI όχι· Μία πιθανή εξήγηση είναι ίσως ότι λόγω των συσχετίσεων που έχουν οι παράμετροι μεταξύ τους, οι ύστερες κατανομές είναι πολυκόρυφες. Το MCMC δεν μπορεί να ξεφύγει από αυτές τις συσχετίσεις και άρα πολλές παραμετρικοποιήσεις του νευρωνικού δικτύου γίνονται αποδεκτές. Από την άλλη χρησιμοποιώντας τον ADVI, λύνουμε ένα πρόβλημα βελτιστοποίησης μη κυρτής συνάρτησης και προσπαθούμε να μοντελοποιήσουμε αυτές τις συσχετίσεις (στην περίπτωση του Full Rank) μέσω των κατανομών που προτείνουμε. Σημαντικό είναι να αναφέρουμε ότι στις προσομοιώσεις που έγιναν με τον ADVI οι κατανομές δεν ήταν μοναδικές. Τρέχοντας το μοντέλο δεύτερη φορά και τρίτη φορά με διαφορετικές αρχικές τιμές, μπορούμε να καταλήξουμε στην ίδια




Γράφημα 2.35: Το διάγραμμα αυτοσυσχέτισης για τα βάρη $\beta_1, \ldots, \beta_H$ αν κρατήσουμε τις τιμές ανά $k = 10$.




Γράφημα 2.36: Οι ύστερες κατανομές για τα βάρη $\boldsymbol{\gamma}$ και τα βάρη $\boldsymbol{\beta}$ και τα αντίστοιχα ιχνογράμματα., όπως προσομοιώθηκαν με τον NUTS για το μοντέλο με πρότερη την τυποποιημένη Κανονική κατανομή, αν κρατήσουμε τις τιμές ανά $k = 10$.

τιμή ελαχίστου του ELBO, αλλά για $\theta$ με διαφορετικές κατανομές. Πρώτα ορίζουμε το μοντέλο στο PyMC3, τρέχουμε την προσομοίωση και στην συνέχεια ορίζουμε για τρίτη φορά πάλι το μοντέλο και ξανατρέχουμε την προσομοίωση.

```
with neural_network:
    inferencefull= pm.FullRankADVI()
    approxfull1= pm.fit(n=200000, method=inferencefull)
tracefull1=approxfull1.sample(2000)
```

```
Average Loss = 111.47: 100%|██████████| 200000/200000 [03:58<00:00, 890.95it/s]
Finished [100%]: Average Loss = 111.47
```

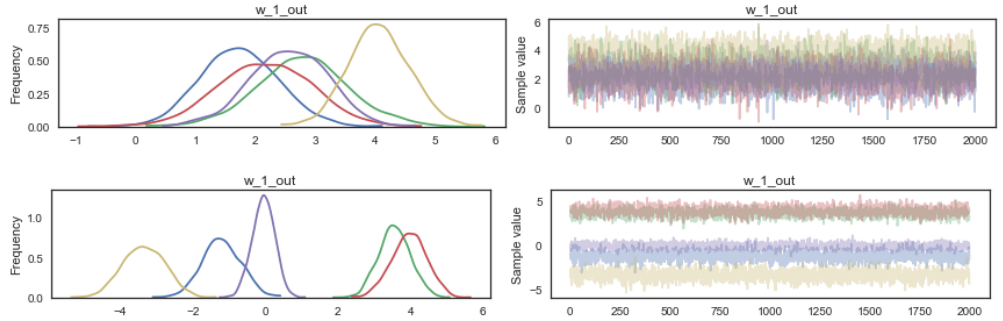
```
with neural_network:
    inferencefull= pm.FullRankADVI()
    approxfull2= pm.fit(n=200000, method=inferencefull)
tracefull2=approxfull2.sample(2000)
```

```
Average Loss = 110.85: 100%|██████████| 200000/200000 [03:40<00:00, 906.29it/s]
Finished [100%]: Average Loss = 110.85
```

Κώδικας 2.37: Στον παραπάνω κώδικα τρέχουμε το ίδιο μοντέλο δυο φορές, ώστε να έχουμε διαφορετικές αρχικές τιμές κάθε φορα.

Ας δούμε για παράδειγμα τις ύστερες κατανομές της μεταβλητής w_1_out. Παρατηρούμε ότι όντως οι ύστερες είναι διαφορετικές.

```
pm.traceplot(tracefull1,varnames=['w_1_out']);
pm.traceplot(tracefull2,varnames=['w_1_out']);
plt.show()
```




Γράφημα 2.38: Οι ύστερες κατανομές για τα βάρη $\beta_1, \ldots, \beta_H$ και τα αντίστοιχα ιχνογράμματα, για το μοντέλο με πρότερη την τυποποιημένη Κανονική κατανομή για διαφορετικές αρχικές τιμές

Γενικά στα νευρωνικά δίκτυα όπου ελαχιστοποιούμε μία συνάρτηση κόστους, υπάρχουν πολλές συμμετρικές λύσεις ακόμα και σε δίκτυα με ένα μόνο κρυφό επίπεδο. Αυτό έχει να κάνει με το ότι η συνάρτηση κόστους δεν είναι κυρτή και άρα έχουμε πολλές τοπικές λύσεις, οι οποίες δεν είναι γενικές. Σύμφωνα με

αυτά, θα μπορούσαμε να υποθέσουμε ότι κάτι τέτοιο συμβαίνει και στα Μπεϋζιανά νευρωνικά δίκτυα όταν χρησιμοποιούμε κάποια προσεγγιστική μέθοδο. Αυτό που κερδίζουμε εδώ είναι μία πιο πρόχειρη λύση, αλλά σε πολύ λιγότερο χρόνο σε σχέση με το πατροπαράδοτο MCMC, το οποίο φαίνεται να είναι πιο ακριβές. Αφού τελειώσαμε και με τον υπολογισμό της ύστερης κατανομής του $\theta$, θα περάσουμε στο στάδιο της πρόβλεψης όπου θα συγκρίνουμε τα παραπάνω μοντέλα μεταξύ τους ως προς την προβλεπτική τους ικανότητα.

## 2.4 Ύστερη Προβλεπτική Κατανομή

Αφού έχουμε χρησιμοποιήσει το δείγμα $D$ για να υπολογίσουμε την ύστερη κατανομή των παραμέτρων $\boldsymbol{\theta}$, μπορούμε να υπολογίσουμε την $Y^\star|\boldsymbol{x}^\star, D$, η οποία είναι η πρόβλεψη του μοντέλου μας για το νέο $\boldsymbol{x}^\star$. Στην Μπεϋζιανή στατιστική αυτό γίνεται μέσω της ύστερης προβλεπτικής κατανομής

$$\begin{aligned} Pr(Y^\star|D) &= \int_{\boldsymbol{\Theta}} Pr(Y^\star|D,\boldsymbol{\theta}) Pr(\boldsymbol{\theta}|D) d\boldsymbol{\theta} \\ &= \int_{\boldsymbol{\Theta}} Pr(Y^\star|\boldsymbol{\theta}) Pr(\boldsymbol{\theta}|D) d\boldsymbol{\theta}. \end{aligned}$$

Ο λόγος που πρέπει να χρησιμοποιήσουμε τον παραπάνω τύπο είναι ότι η παράμετρος $\boldsymbol{\theta}$ δεν είναι σταθερή (όπως στην κλασσική στατιστική), αλλά ακολουθεί πλέον κάποια κατανομή. Έτσι για να υπολογίσουμε την $Pr(Y^\star|D)$ θα πρέπει να ολοκληρώσουμε πάνω σε όλο τον παραμετρικό χώρο $\boldsymbol{\Theta}$ την $Pr(Y^\star|D,\boldsymbol{\theta})$ λαμβάνοντας υπόψιν και την κατανομή του $\theta$, η οποία είναι η ύστερη κατανομή $Pr(\boldsymbol{\theta}|D)$ που υπολογίσαμε στο προηγούμενο κεφάλαιο. Στο PyMC3 η ύστερη προβλεπτική κατανομή υλοποιείται μέσω της συνάρτησης pm.sample_ppc(). Αυτή η συνάρτηση παίρνει ως ορίσματα τις τιμές που προσομοιώσαμε, δηλαδή το αντικείμενο τύπου trace, το μοντέλο από το οποίο προέρχονται και τον αριθμό των δειγμάτων που θέλουμε να παραγάγουμε. Το τελευταίο όρισμα έχει να κάνει με το ότι για κάθε καινούργια πρόβλεψη του μοντέλου έχουμε μία ολόκληρη κατανομή και όχι απλά μία τιμή. Έτσι για να εκτιμήσουμε αυτήν την κατανομή αρκεί να παράγουμε πολλά δείγματα από αυτήν. Στην συνέχεια για να έχουμε μία αντιπροσωπευτική τιμή για κάθε πρόβλεψη, μπορούμε να πάρουμε την μέση τιμή της ύστερης προβλεπτικής κατανομής και χρησιμοποιώντας αυτήν να ελέγξουμε την προβλεπτική ικανότητα του μοντέλου βάση κάποιας μετρικής. Επειδή έχουμε να κάνουμε με πρόβλημα κατηγοριοποίησης και το μοντέλο μας επιστρέφει την πιθανότητα η $Pr(Y^\star = 1|\boldsymbol{x}^\star)$ , η μετρική που θα χρησιμοποιήσουμε είναι η ακρίβεια του μοντέλου μας, η οποία είναι ο λόγος του αριθμού των τιμών που προβλέψαμε σωστά δια τον συνολικό αριθμό των τιμών που προβλέψαμε. Χωρίς βλάβη της γενικότητας θέτουμε $Y^\star = 1$ αν ο μέσος

της ύστερης προβλεπτικής κατανομής είναι πάνω από 0.5 και $Y^\star = 0$ διαφορετικά. Ξεκινάμε ορίζοντας μία συνάρτηση με όνομα prediction_accuracy() και ορίσματα αυτά που χρειάζεται η pm.sample_ppc() για να τρέξει.

```
def prediction_accuracy(trace,model,samples):
    # Replace shared variables with testing set
    ann_input.set_value(X_test)
    # Same procedure for Test set
    ppc = pm.sample_ppc(trace, model=model, samples=samples)
    # Create a 500 x 152 with prediction for the Test set
    pred = ppc['out'].mean(axis=0) > 0.5
    print('Test Accuracy = {}%'.format((Y_test == pred).mean() * 100))
    return pred
```

Κώδικας 2.39: Ο κώδικας για την συνάρτηση prediction_accuracy()

Θυμίζουμε ότι το αποτέλεσμα του μοντέλου που έχουμε δημιουργήσει έχει όνομα out. Εκχωρούμε το αποτέλεσμα της pm.sample_ppc() σε μία μεταβλητή με όνομα ppc. Η μεταβλητή αυτή είναι ένα πλαίσιο δεδομένων $n \times samples$, όπου $n$ ο αριθμός των προβλέψεων που θέλουμε να κάνουμε. Με την εντολή $ppc['out'].mean(axis = 0)$ παίρνουμε ένα διάνυσμα διάστασης $n$ το οποίο είναι ο μέσος της ύστερης προβλεπτικής κατανομής. Στην συνέχεια ελέγχουμε για κάθε $j = 1, \ldots, n$ αν αυτός ο μέσος είναι μεγαλύτερος από 0.5 ή όχι για να δώσουμε την τιμή 1 ή 0 σε κάθε $y_j^\star$. Το αποτέλεσμα εκχωρείται σε μία μεταβλητή με όνομα pred, το οποίο είναι και το διάνυσμα με τις προβλέψεις μας. Η συνάρτηση υλοποιεί την παραπάνω διαδικασία για τα δεδομένα ελέγχου X_test. Βέβαια για να χρησιμοποιήσουμε τα δεδομένα X_test, πρέπει να αλλάξουμε την τιμή της μεταβλητής ann_input χρησιμοποιώντας την προέκταση .set_value(). Έτσι θα μπορεί η pm.sample_ppc() να τρέξει το μοντέλο για τα καινούργια δεδομένα. Στο σύνολο έχουμε δημιουργήσει τρία μοντέλα νευρωνικών δικτύων με διαφορετικές πρότερες κατανομές, στα οποία χρησιμοποιήσαμε τον αλγόριθμο ADVI για να υπολογίσουμε την ύστερη. Επίσης στο ένα από αυτά τα μοντέλα (τυποποιημένη πρότερη κατανομή) υπολογίσαμε την ύστερη με τον MCMC αλγόριθμο NUTS και βεβαιωθήκαμε ότι η σύγκλιση είχε επιτευχθεί. Σκοπός μας τώρα είναι χρησιμοποιώντας την συνάρτηση prediction_accuracy() να δούμε την προβλεπτική ικανότητα των παραπάνω μοντέλων. Το πρώτο μοντέλο είχε για πρότερη κατανομή την flat και είχε όνομα neural_network_flat. Ξεκινάμε παράγοντας 500 τιμές για κάθε ύστερη προβλεπτική κατανομή. Όπως βλέπουμε (Κώδικας 2.40) η δειγματοληψία γίνεται αρκετά γρήγορα σε λιγότερο από 10 δευτερόλεπτα. Η προβλεπτική ικανότητα του πρώτου μοντέλου είναι 96%, κάτι το οποίο είναι αρκετά ικανοποιητικό. Συνεχίζουμε με το μοντέλο με πρότερη κατανομή την τυποποιημένη Κανονική και το Ιεραρχικό μοντέλο.

```
pred=prediction_accuracy(tracefull,neural_network_flat,500)
100%|| 500/500 [00:05<00:00, 85.55it/s]
Test Accuracy = 96.0%
```

Κώδικας 2.40: Η προβλεπτική ικανότητα του μοντέλου με την flat πρότερη.

```
pred=prediction_accuracy(tracefull,neural_network,500)
100%|| 500/500 [00:00<00:00, 1110.26it/s]
Test Accuracy = 95.8%
```

Κώδικας 2.41: Η προβλεπτική ικανότητα του μοντέλου με την τυποποιημένη Κανονική πρότερη.

```
pred=prediction_accuracy(tracefull,neural_network_h,500)
100%|| 500/500 [00:00<00:00, 1184.66it/s]
Test Accuracy = 96.39999999999999%
```

Κώδικας 2.42: Η προβλεπτική ικανότητα του Ιεραρχικού μοντέλου.

Παρατηρούμε ότι και τα τρία μοντέλα έχουν περίπου την ίδια προβλεπτική ικανότητα στα δεδομένα ελέγχου. Επίσης βλέπουμε ότι μέχρι στιγμής όλα τα νευρωνικά μας δίκτυα προβλέπουν καλύτερα από το λογιστικό μοντέλο (86%) που χρησιμοποιήσαμε στην αρχή του Κεφαλαίου. Το τελευταίο μοντέλο που θα δούμε, είναι αυτό που προσαρμόσαμε με τον NUTS. Οι τιμές που γεννήσαμε είναι αποθηκευμένες στην μεταβλητή trace_NUTS2.

```
pred=prediction_accuracy(trace_NUTS2,neural_network,500)
100%|| 500/500 [00:00<00:00, 1172.56it/s]
Test Accuracy = 96.2%
```

Κώδικας 2.43: Η προβλεπτική ικανότητα του μοντέλου με την τυποποιημένη Κανονική πρότερη(NUTS).

Τα αποτελέσματα και σε αυτήν την περίπτωση είναι αρκετά καλά με προβλεπτική ικανότητα σχεδόν ίδια με αυτήν του ADVI στο 96%. Έτσι καταλήγουμε στο συμπέρασμα ότι και τα τέσσερα μοντέλα μας φαίνονται να έχουν τις ίδιες ικανότητες όσο αναφορά την πρόβλεψη. Κλείνοντας το κεφάλαιο, θα δούμε πως προσαρμόζεται το μοντέλο μας σε ένα υποσύνολο του $\mathbb{R}^2$. Λαμβάνοντας υπόψιν το εύρος της επεξηγηματικής μεταβλητής, θα προβλέψουμε την τιμή $Y^\star$ για κάθε $\boldsymbol{x}^\star \in [-3,3]^2$ χρησιμοποιώντας ένα από τα παραπάνω μοντέλα. Για λόγους ευκολίας θα χρησιμοποιήσουμε το μοντέλο με την τυποποιημένη Κανονική πρότερη κατανομή που προσαρμόσαμε με τον ADVI.

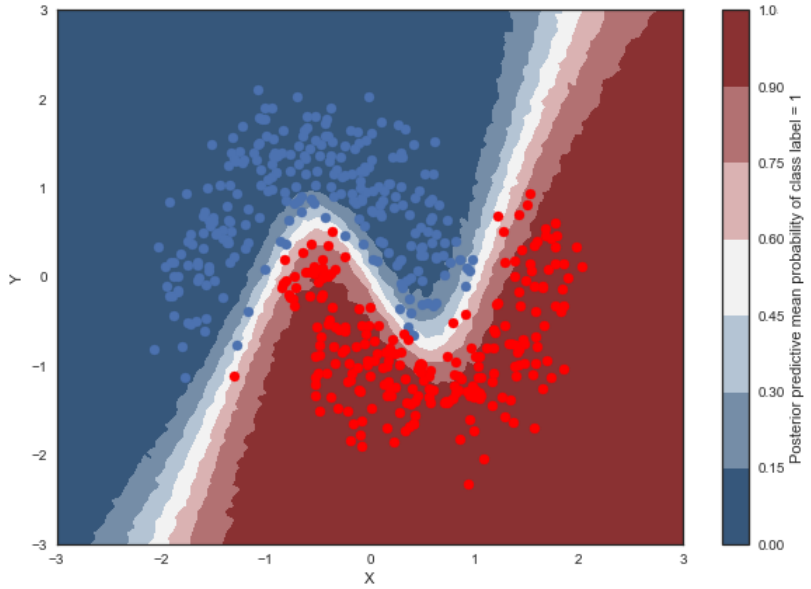
Επειδή τα σημεία του $[-3,3]^2$ είναι άπειρα, θα δημιουργούμε ένα πλέγμα (grid) από το -3 εώς το 3 για την επεξηγηματική μεταβλητή $\boldsymbol{X}^\star = \boldsymbol{x}^\star$, έτσι ώστε να έχουμε σύνολο 10000 σημεία για την $\boldsymbol{X}^\star$ και να προβλέψουμε την αντίστοιχη τιμή της $Y^\star$ για όλα αυτά. Για να το κάνουμε αυτό θα χρησιμοποιήσουμε την συνάρτηση np.mgrid() για να δημιουργήσουμε ένα διάνυσμα διάστασης (2,100,100) και στην συνέχεια το μετασχηματίζουμε χρησιμοποιώντας την προέκταση .reshape() για να φτιάξουμε το ορθογώνιο που θέλουμε. Αλλάζουμε την τιμή της μεταβλητής εισόδου ann_input στο παραπάνω ορθογώνιο και ξεκινάμε την πρόβλεψη για τα 10000 σημεία.

```
grid = np.mgrid[-3:3:100j,-3:3:100j]
grid_2d = grid.reshape(2, -1).T
X, Y = grid
ann_input.set_value(grid_2d)
ppc = pm.sample_ppc(trace=tracefull,model=neural_network, samples=500)
```

Κώδικας 2.44: Ο κώδικας για να κατασκεύασουμε το πλέγμα του $[-3,3]^2$ και να προβλέψουμε την κατηγορία στη οποία ανήκουν όλα τα σημεία του πλέγματος.

Αφού πρώτα προσομοιώσαμε 500 τιμές από την ύστερη προβλεπτική κατανομή για κάθε ένα σημείο του πλέγματος, συνεχίζουμε δημιουργώντας ένα γράφημα με τον μέσο αυτών των τιμών. Ορίζουμε μία μεταβλητή με όνομα cmap η οποία δηλώνει ποια παλέτα χρωμάτων θα χρησιμοποιήσουμε για να δημιουργήσουμε ένα διάγραμμα ισοϋψών καμπυλών contourplot. Με την προέκταση .contourf() χωρίζουμε τον μέσο της ύστερης σε 7 διαφορετικές κατηγορίες και χρωματίζουμε με διαφορετικό χρώμα την κάθε κατηγορία. Τέλος προσθέτουμε και τις προβλέψεις του συνόλου ελέγχου X_test πάνω στο γράφημα. Το αποτέλεσμα φαίνονται στο Γράφημα 2.45. Η λευκή γραμμή αντιστοιχεί στον μέσο της ύστερης προβλεπτικής κατανομή που κυμαίνεται από 0.45 έως 0.6. Αλλιώς, θα μπορούσαμε να το χαρακτηρίσουμε και ως το σύνορο όπου διαχωρίζει τα σημεία ανάλογα με ποια κλάση ανήκουν, αφού επιλέξαμε ως πιθανότητα διαχωρισμού την τιμή 0.5. Ομοίως μπορούμε να περιγράψουμε και τα άλλα χρώματα. Θα μπορούσαμε να πούμε ότι το μοντέλο μας κατάφερε να αναγνωρίσει αυτό το δύσκολο μοτίβο μετά τα μισοφέγγαρα. Πολλές φορές θα θέλαμε για μία μελλοντική πρόβλεψη που κάνει το μοντέλο, να ξέραμε και με πόση βεβαιότητα έγινε. Αυτήν την εκτίμηση μπορούμε να την πάρουμε μέσω της τυπικής απόκλισης της ύστερης προβλεπτικής κατανομής, η οποία δηλώνει την αβεβαιότητα της για κάθε πρόβλεψη. Για να το καταλάβουμε αυτό λίγο καλύτερα μπορούμε να δημιουργήσουμε και ένα γράφημα με την τυπική απόκλιση της ύστερης προβλεπτικής κατανομής της παραμέτρου $p$. Πρώτα θα εκτιμήσουμε την τυπική απόκλιση από το δείγμα μας $ppc['out']$ χρησιμοποιώντας την προέκταση $.std(axis = 0)$ και στη συνέχεια θα δημιουργήσουμε ένα γράφημα ισοϋψών καμπυλών όπως και με το μέσο. Τα αποτελέσματα φαίνονται στο Γράφημα 2.46.

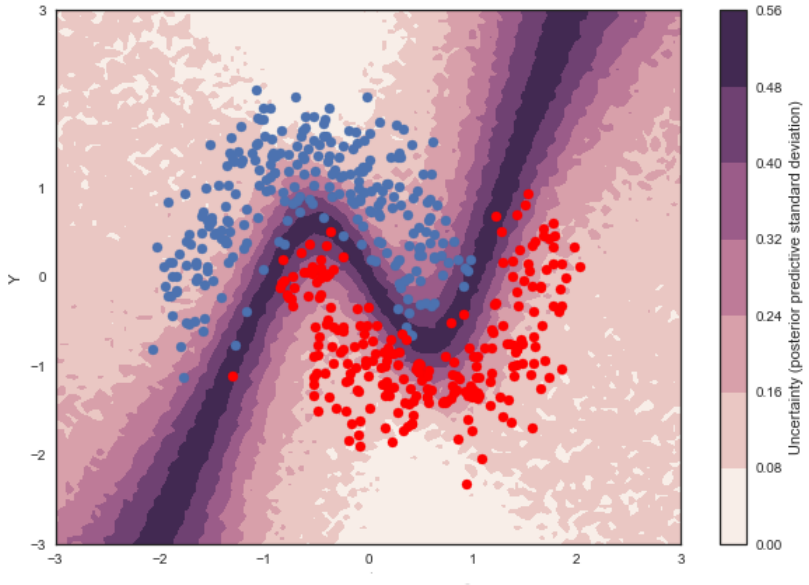
```
cmap = sns.diverging_palette(250, 12, s=85, l=25, as_cmap=True)
fig, ax = plt.subplots(figsize=(10, 7))
contour = ax.contourf(X, Y, ppc['out'].mean(axis=0).reshape(100, 100), cmap=cmap)
ax.scatter(X_test[pred==0, 0], X_test[pred==0, 1])
ax.scatter(X_test[pred==1, 0], X_test[pred==1, 1], color='r')
cbar = plt.colorbar(contour, ax=ax)
_ = ax.set(xlim=(-3, 3), ylim=(-3, 3), xlabel='X', ylabel='Y');
cbar.ax.set_ylabel('Posterior predictive mean probability of class label = 1');
```




Γράφημα 2.45: Στην παραπάνω εικόνα βλεπουμε την κατηγορία που προέβλεψε το μοντέλο μας για τα δεδομένα ελέγχου, αλλά και πώς κατανέμεται ο μέσος τη ύστερης κατανομής του $p$ για $\boldsymbol{x}^\star \in [-3, 3]^2$.

Με το μοβ χρώμα έχουμε την μεγαλύτερη αβεβαιότητα, η οποία κυμαίνεται από το 0.48 εώς το 0.56. Προφανώς το σύνορο των κλάσεων θα πρέπει να έχει την μεγαλύτερη αβεβαιότητα για ένα μοντέλο, ενώ όσο απομακρυνόμαστε από αυτό, η αβεβαιότητα πρέπει να τείνει στο μηδεν. Επίσης, χαρακτηριστικά φαίνονται και οι πιο έντονες εναλλαγές των χρωμάτων στα σημεία του χώρου όπου το μοντέλο δεν έχει προσαρμοστεί. Συνδυάζοντας τα προηγούμενα γραφήματα μαζί, μπορούμε να έχουμε μία καλύτερη εκτίμηση για τον τρόπο με τον οποίο κατανέμονται οι πιθανότητες του μοντέλου που δημιουργήσαμε για κάθε σημειο του χώρου.

```
cmap = sns.cubehelix_palette(light=1, as_cmap=True)
fig, ax = plt.subplots(figsize=(10, 7))
contour = ax.contourf(X, Y, ppc['out'].std(axis=0).reshape(100, 100), cmap=cmap)
ax.scatter(X_test[pred==0, 0], X_test[pred==0, 1])
ax.scatter(X_test[pred==1, 0], X_test[pred==1, 1], color='r')
cbar = plt.colorbar(contour, ax=ax)
_ = ax.set(xlim=(-3, 3), ylim=(-3, 3), xlabel='X', ylabel='Y');
cbar.ax.set_ylabel('Uncertainty (posterior predictive standard deviation)');
```




Γράφημα 2.46: Στην παραπάνω εικόνα βλεπουμε πάλι την κατηγορία που προέβλεψε το μοντέλο μας για τα δεδομένα ελέγχου, αλλά και πώς κατανέμεται η τυπική απόκλιση τη ύστερης κατανομής του $p$ για $\boldsymbol{x}^\star \in [-3, 3]^2$.

# Κεφάλαιο 3

# Δεδομένα Boston: Πρόβλεψη Τιμής Σπιτιών

Στο τελευταίο κεφάλαιο αυτής της εργασίας θα μοντελοποιήσουμε ένα πρόβλημα παλινδρόμησης με ένα Μπεϋζιανό νευρωνικό δίκτυο. Για αυτό το πρόβλημα θα επιλέξουμε ένα πραγματικό σύνολο δεδομένων που αφορά τα σπίτια στην Βοστόνη των Η.Π.Α. (The Boston Housing Dataset), το οποίο βρίσκεται στην ιστοσελίδα http://www.cs.toronto.edu/ delve/data/boston/bostonDetail.html. Τα δεδομένα που θα εξετάσουμε αποτελούνται από 14 μεταβλητές στο σύνολο, με 506 παρατηρήσεις η κάθε μία. Αυτές οι μεταβλητές αφορούν διάφορα χαρακτηριστικά ενός σπιτιού, όπως π.χ. την τιμή του σε 1000$, ο αριθμός των δωματίων, το επίπεδο εγκληματικότητας κ.α. Οι 13 μεταβλητές είναι συνεχής, ενώ η μεταβλητή CHAS είναι κατηγορική με δύο κατηγορίες. Συγκεκριμένα θα επιλέξουμε την μεταβλητή που αφορά την τιμή των σπιτιών ως μεταβλητή απόκρισης $Y$ και τις υπόλοιπες ως επεξηγηματικές μεταβλητές $\boldsymbol{X} = (X_0, \ldots, X_{12})$. Πρώτα φορτώνουμε τις βιβλιοθήκες που θα χρησιμοποιήσουμε στην Python. Εκτός από αυτές που χρησιμοποιήσαμε στο Κεφάλαιο 2, θα χρειαστούμε ακόμα την pandas για να δημιουργήσουμε διάφορα πλαίσια δεδομένων για περιγραφική στατιστική και την sklearn.datasets για να κατεβάσουμε τα δεδομένα Boston. Στην συνέχεια δημιουργούμε ένα πλαίσιο δεδομένων με μία σύνοψη των βασικών χαρακτηριστικών των μεταβλητών μας αφαιρώντας την μεταβλητή CHAS η οποία είναι κατηγορική (Πινακάκι 3.3). Η μεταβλητή απόκρισης $Y$ είναι η μεταβλητή MEDV και είναι η διάμεσος των τιμών (σε 1000$) που προτάθηκαν για το συγκεκριμένο σπίτι.

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
import pymc3 as pm
import theano.tensor as T
import theano
import sklearn as sk
from pymc3 import *
import seaborn as sns
sns.set_style('white')
import matplotlib.pyplot as plt
from pandas import DataFrame
```

Κώδικας 3.1: Ο κώδικας για να φορτώσουμε τις βιβλιοθήκες της Python που θα χρησιμοποιήσουμε.

| | |
|---|---|
| **CRIM** | **Ποσοστό εγκληματικότητας κατά κεφαλή** |
| **ZN** | **Ποσοστό κατοικήσιμης γης πάνω από 25000 τ.πόδια** |
| **INDUS** | **Ποσοστό επιχειρήσεων παροχής υπηρεσιών** |
| **CHAS** | **Ψευδομεταβλητή του Charles (1 αν το σπίτι είναι κοντά σε ποτάμι, 0 διαφορετικά** |
| **NOX** | **Συγκέντρωση Μονοξειδίου του αζώτου** |
| **RM** | **Μέσος αριθμός δωματίων ανα κατοικία** |
| **AGE** | **Ποσοστό των ιδιόκτητων κατοικιών που χτίστηκαν πριν το 1940** |
| **DIS** | **Σταθμισμένη απόσταση από πέντε Κέντρα Εργασίας στη Βοστώνη** |
| **RAD** | **Δείκτης πρόσβασης σε κεντρικές λεωφόρους** |
| **TAX** | **Φόρος ιδιοκτησίας ανά εισόδημα 10.000$** |
| **PTRATIO** | **Αναλογία μαθητών-δασκάλων ανά πόλη** |
| **B** | **1000(Bk -0.63)^2, όπου Bk είναι το ποσοστό των έγχρομων ανα πόλη** |
| **LSTAT** | **Ποσοστό της χαμηλής τάξης του πληθυσμού** |
| **MEDV** | **Διάμεσος της τιμής των σπιτιών σε 1000$** |

Πινακάκι 3.2: Η περιγραφή των μεταβλητών που είναι διαθέσιμες στο σύνολο δεδομένων Boston.

```
from pandas import DataFrame
dfX=DataFrame(X)
dfX.columns = ['CRIM','ZN','INDUS','CHAS','NOX','RM',
               'AGE','DIS','RAD','TAX','PTRATIO','B','LSTAT']
dfX['MEDV']=Y
dfX.drop('CHAS', 1).describe().transpose()
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **CRIM** | 506.0 | 3.593761 | 8.596783 | 0.00632 | 0.082045 | 0.25651 | 3.647423 | 88.9762 |
| **ZN** | 506.0 | 11.363636 | 23.322453 | 0.00000 | 0.000000 | 0.00000 | 12.500000 | 100.0000 |
| **INDUS** | 506.0 | 11.136779 | 6.860353 | 0.46000 | 5.190000 | 9.69000 | 18.100000 | 27.7400 |
| **NOX** | 506.0 | 0.554695 | 0.115878 | 0.38500 | 0.449000 | 0.53800 | 0.624000 | 0.8710 |
| **RM** | 506.0 | 6.284634 | 0.702617 | 3.56100 | 5.885500 | 6.20850 | 6.623500 | 8.7800 |
| **AGE** | 506.0 | 68.574901 | 28.148861 | 2.90000 | 45.025000 | 77.50000 | 94.075000 | 100.0000 |
| **DIS** | 506.0 | 3.795043 | 2.105710 | 1.12960 | 2.100175 | 3.20745 | 5.188425 | 12.1265 |
| **RAD** | 506.0 | 9.549407 | 8.707259 | 1.00000 | 4.000000 | 5.00000 | 24.000000 | 24.0000 |
| **TAX** | 506.0 | 408.237154 | 168.537116 | 187.00000 | 279.000000 | 330.00000 | 666.000000 | 711.0000 |
| **PTRATIO** | 506.0 | 18.455534 | 2.164946 | 12.60000 | 17.400000 | 19.05000 | 20.200000 | 22.0000 |
| **B** | 506.0 | 356.674032 | 91.294864 | 0.32000 | 375.377500 | 391.44000 | 396.225000 | 396.9000 |
| **LSTAT** | 506.0 | 12.653063 | 7.141062 | 1.73000 | 6.950000 | 11.36000 | 16.955000 | 37.9700 |
| **MEDV** | 506.0 | 22.532806 | 9.197104 | 5.00000 | 17.025000 | 21.20000 | 25.000000 | 50.0000 |

Πινακάκι 3.3: Τα βασικά χαρακτηριστικά των μεταβλητών μας, χωρίς την μεταβλητή CHAS

Όπως παρατηρούμε, οι μεταβλητές μας κυμαίνονται σε διαφορετικό εύρος και άρα πριν τις χωρίσουμε σε δύο ξένα υποσύνολα, θα πρέπει να τις μετασχηματίσουμε αφαιρώντας το μέσο και διαιρώντας με την τυπική απόκλιση. Όπως και στο Κεφάλαιο 2, θα χρησιμοποιήσουμε την συνάρτηση scale(). Τέλος με την προέκταση .shape μπορούμε να δούμε την διάσταση των δύο υποσυνόλων.

```
X=boston.data
#Scale the independent variable
XX=sk.preprocessing.scale(X)
#Replace the binary variable "CHAS"
XX[:,3]=X[:,3]
#Set the target variable,split the data and check their size
Y=boston.target
X_train, X_test, Y_train, Y_test = train_test_split(XX, Y, test_size=.3)
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
```

```
((354L, 13L), (354L,))
((152L, 13L), (152L,))
```

Κώδικας 3.4: Ο κώδικας για να κανονικοποιήσουμε τα δεδομένα και να τα χωρίσουμε σε δύο σύνολα δεδομένων ($Train\_set$ 70% και $Test\_set$ 30% )

Αφού ετοιμάσαμε κατάλληλα τα δεδομένα μπορούμε να ξεκινήσουμε την ανάλυσή μας. Σκοπός μας είναι να δημιουργήσουμε ένα νευρωνικό δίκτυο, το οποίο να προβλέπει αρκετά καλά την τιμή των σπιτιών στη Βοστόνη. Για να δούμε την προβλεπτική ικανότητα των μοντέλων μας και να επιλέξουμε το καλύτερο από αυτά, θα χρησιμοποιήσουμε 3 διαφορετικές μετρικές. Η πρώτη είναι το Μέσο Τετραγωνικό Σφάλμα (Mean Squared Error) που είδαμε στο Κεφάλαιο 1, η δεύτερη το Μέσο Απόλυτο Σφάλμα (Mean Absolute Error) και τελευταία η αρνητική πιθανοφάνεια του μοντέλου σε λογαριθμική κλίμακα (Mean Negative Log Probability Density). Αν $n$ είναι ο αριθμός των προβλέψεων $Y^\star$ σε ένα σύνολο δεδομένων, τότε οι παραπάνω μετρικές ορίζονται ως εξής:

$$MSE = \sum_{i=1}^{n} \frac{(y_i - y_i^\star)^2}{n}$$

$$MAE = \sum_{i=1}^{n} \frac{|y_i - y_i^\star|}{n}$$

$$NLPD = \sum_{i=1}^{n} \left( \frac{1}{2} log(2\pi\sigma_{\star i}^2) + \frac{(y_i - y_i^\star)^2}{2\sigma_{\star i}^2} \right),$$

όπου $\sigma_{\star i} = \mathbb{V}(f_{\star i}) + \sigma$. Η ποσότητα $\mathbb{V}(f_{\star i})$ εκτιμάτε από την τυπική απόκλιση της ύστερης προβλεπτικής κατανομής για την πρόβλεψη $Y_i^\star$ και $\sigma$ είναι η τυπική απόκλιση της παλινδρόμησης, δηλαδή η τυπική απόκλιση των σφαλμάτων $\varepsilon$. Επειδή το $\sigma$ αποτελεί παράμετρο στο μοντέλο μας θα χρησιμοποιήσουμε την μέση

τιμή της ύστερης κατανομής του για να εκτιμήσουμε την παραπάνω ποσότητα. Δημιουργούμε πρώτα την συνάρτηση prediction_accuracy, η οποία είναι ίδια με αυτήν που χρησιμοποιήσαμε στο Κεφάλαιο 2 με την διαφορά ότι μας εμφανίζει τις παραπάνω μετρικές που είναι κατάλληλες για προβλήματα παλινδρόμησης.

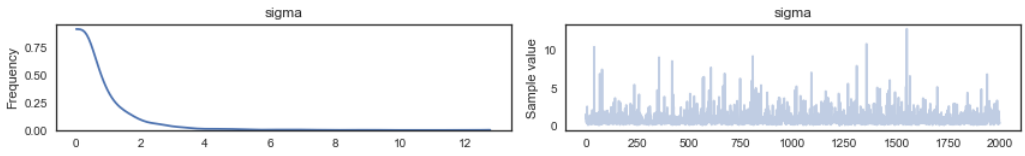
```
def prediction_accuracy(trace,model,samples):
    # Replace shared variables with testing set
    ann_input.set_value(X_test)
    # Same procedure for Test set
    ppc = pm.sample_ppc(trace, model=model, samples=samples)
    # Create a 500 x 152 with prediction for the Test set
    pred = ppc['Y_obs']
    NLPD = 0.5*np.log(2*np.pi*(pred.std(axis=0)**2)) + (((pred.mean(axis=0)-Y_test)**2)/(2*sigma_mean**2))
    MSE = sum((pred.mean(axis=0)-Y_test)**2)/Y_test.shape
    MAE = sum(abs(pred.mean(axis=0)-Y_test))/Y_test.shape
    print('Test MSE = {}'.format(MSE))
    print('Test MAE = {}'.format(MAE))
    print('Train NLPD = {0:.3f}'.format(NLPD.mean()))
    return pred,pred2
```

Κώδικας 3.5: Ο κώδικας της συνάρτησης prediction_accuracy για την παλινδρόμηση.

Όπως είδαμε στο Κεφάλαιο 1, ένα μοντέλο παλινδρόμησης έχει πάντα τουλάχιστον μία παραπάνω παράμετρο, την διασπορά του μοντέλου $\sigma$. Αυτή η διασπορά προέρχεται από την υπόθεση που κάνουμε ότι $\varepsilon_i \sim N(0, \sigma^2)$. Η πρότερη που θα δώσουμε στο $\sigma$ θα είναι περικομμένη στο μηδέν Κανονική κατανομή με παράμετρο θέσης τη μονάδα (Standardizes Half Normal Distribution). Χρησιμοποιώντας το PyMC3, προσομοιώνουμε τιμές με τον ADVI για να δούμε την μορφή της συγκεκριμένης κατανομής (Γράφημα 3.6).

```
with pm.Model() as Hnormal:
    sigma=pm.HalfNormal('sigma', sd=1)
with Hnormal:
    iinference = pm.FullRankADVI()
    aapproxfull = pm.fit(n=20000, method=iinference)
trace=aapproxfull.sample(2000)
pm.traceplot(trace);
```

```
Average Loss = 0.55605: 100%|██████████| 20000/20000 [00:08<00:00, 2372.51it/s]
Finished [100%]: Average Loss = 0.55846
```

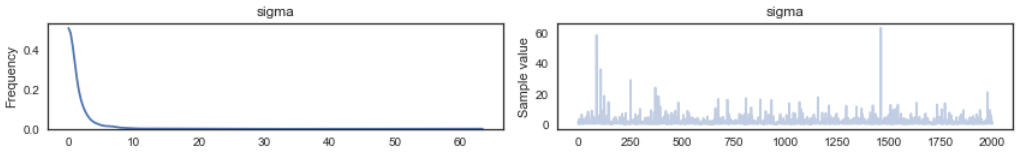



Γράφημα 3.6: Η περικομμένη στο μηδέν Κανονική κατανομή με παράμετρο θέσης τη μονάδα, όπως προσομοιώθηκε από τον ADVI.

Αυτή η πρότερη έχει μέση τιμή $\frac{\sqrt{2}}{\sqrt{\pi}} \simeq 0.8$ και είναι αρκετά πληροφοριακή. Η επιλογή αυτή στηρίζεται στην πεποίθησή μας ότι η διασπορά των σφαλμάτων του μοντέλου είναι αρκετά μικρή. Εκτός από την παραπάνω κατανομή δοκιμάσαμε μία flat και μία περικομένη στο μηδέν t-Student κατανομή με βαθμούς ελευθερίας $\nu = 2$.

```
with pm.Model() as HStudent:
    sigma=pm.HalfStudentT('sigma',nu=2)
with HStudent:
    iinference = pm.FullRankADVI()
    aapproxfull = pm.fit(n=20000, method=iinference)
trace=aapproxfull.sample(2000)
pm.traceplot(trace);
```

```
Average Loss = 0.74722: 100%|██████████| 20000/20000 [00:07<00:00, 2732.28it/s]
Finished [100%]: Average Loss = 0.74626
```




Γράφημα 3.7: Η περικομμένη στο μηδέν t-Student κατανομή με βαθμούς ελευθερίας $\nu = 2$, όπως προσομοιώθηκε από τον ADVI.

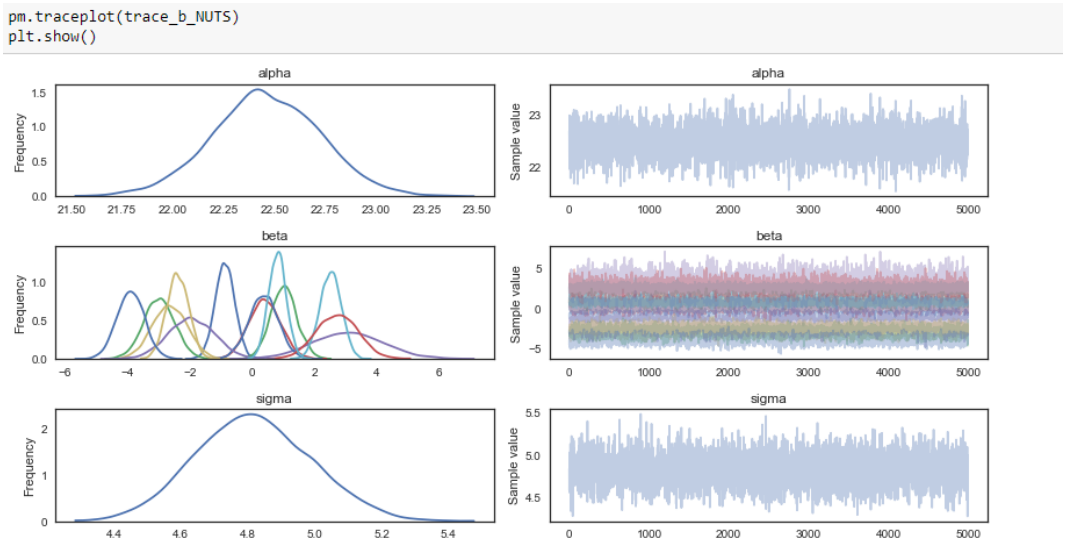
Τα τελικά αποτελέσματα, όπως θα δούμε και παρακάτω, ήταν σχεδόν ίδια για τις παραπάνω πρότερες, που σημαίνει ότι δεδομένα ήταν αρκετά πιο ισχυρά από τις πεποιθήσεις μας, ώστε να μετατοπίσουν την ύστερη κατανομή του $\sigma$ στην ίδια μορφή ανεξάρτητα από την πρότερη επιλογή. Το πρώτο μοντέλο που χρησιμοποιήσαμε ήταν ενα γραμμικό μοντέλο με πρότερες Κανονικές κατανομές με μέση τιμή 0 και διασπορά $100^2$ για τις 14 παραμέτρους των συντελεστών, ενώ για στην διασπορά $\sigma$ δώσαμε την περικομμένη τυποποιημένη Κανονική. Υπολογίσαμε την ύστερη και με τον ADVI και με τον NUTS και είδαμε ότι τα αποτελέσματα ήταν ακριβώς ίδια. Αυτό επιβεβαιώνει ότι και η μία, αλλά και η άλλη μέθοδος δούλεψαν πολύ καλά και βρήκαν την πραγματική ύστερη κατανομή των παραμέτρων.

```
prediction_accuracy(trace_b_NUTS,basic_model,500)
```

```
100%|██████████| 500/500 [00:00<00:00, 1108.92it/s]
Test MSE = [ 18.1929242]
Test MAE = [ 3.25300168]
Train NLPD = 2.904
```

Κώδικας 3.8: Η προβλεπτική ικανότητα του γραμμικού μοντέλου.

Το σφάλμα το μοντέλου φαίνεται λίγο υψηλό (σε σχέση με τα αποτελέσματα των νευρωνικών δικτύων που θα δούμε παρακάτω), αλλά για ένα τόσο απλό μοντέλο τα αποτελέσματα φαίνονται ικανοποιητικά.




Γράφημα 3.9: Το γράφημα των ύστερων κατανομών των παραμέτρων μας και τα αντίστοιχα ιχνογράμματα για το γραμμικό μοντέλο, όπως προσομοιώθηκαν από τον NUTS.

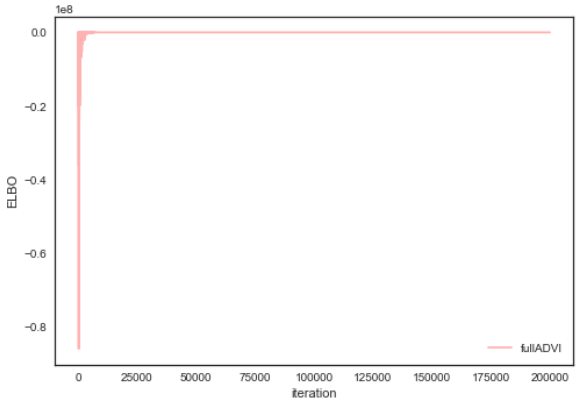
Τα ιχνόγράμματα φαίνονται πάρα πολύ καλά για το MCMC. Βέβαια ο αριθμός των παραμέτρων στο γραμμικό μοντέλο, είναι σχετικά μικρός σε σχέση με ένα νευρωνικό δίκτυο που είναι αρκετά πιο πολύπλοκο. Στην συνέχεια δοκιμάσαμε ένα νευρωνικό δίκτυο με ένα κρυφό επίπεδο με 5 νευρώνες για διαφορετικές πρότερες κατανομές. Ο αριθμός των νευρώνων στο κρυφό επίπεδο επιλέχθηκε πάλι βάσει της προβλεπτικής ικανότητας του μοντέλου. Για συνάρτηση ενεργοποίησης επιλέξαμε την ReLU, γιατί είχε καλύτερη προβλεπτική ικανότητα από την tanh. Για τους νευρώνες του δικτύου οι πρότερες ήταν μία flat, μία τυποποιημένη Κανονική, μία ARD και ένα Ιεραρχικό μοντέλο πρώτης τάξης. Η διαφορά του ARD με το Ιεραρχικό μοντέλο που είδαμε στο Κεφάλαιο 2 είναι ότι είναι ότι έχει δύο επίπεδα υπερπαραμέτρων. Για την μοντελοποίηση του ARD στο PyMC3 χρησιμοποιήσαμε μία πολυδιάστατη Κανονική κατανομή με διαγώνιο $\Sigma^{-1}$ και διαγώνια στοιχεία την ακρίβεια $\tau_i \sim Gamma(a = 0.1, b = \frac{0.1}{\tau_\star})$ με $\tau_\star \sim Gamma(a = 0.1, b = 0.001)$. Η ακρίβεια $\tau = \frac{1}{\sigma^2}$ συνηθίζεται να χρησιμοποιείται αντί της διασποράς $\sigma^2$ για υπολογιστικούς σκοπούς. Ουσιαστικά η υπερπαράμετρος $\tau_\star$ στο πρώτο επίπεδο ελέγχει την $\tau_i$ στο δεύτερο επίπεδο, η οποία με τη σειρά της ελέγχει την διασπορά της Κανονικής πρότερης για τα βάρη $\gamma$. Έτσι αν η διασπορά για κάποιο βάρος $\gamma_i$ είναι αρκετά μικρή, τότε η αντίστοιχη μεταβλητή $X_i$ θα αγνοηθεί από το μοντέλο. Αρχικά για τον υπολογισμό της ύστερης διαλέξαμε τον ADVI, λόγω του μεγάλου αριθμού των παραμέτρων.

Τρέξαμε τον αλγόριθμο για 200000 επαναλήψεις και στην συνέχεια προσομοιώσαμε 2000 τιμές από κάθε ύστερη κατανομή. Η σύγκλιση του αλγορίθμου επιτεύχθηκε πολύ γρήγορα ανεξάρτητα από την πρότερη που χρησιμοποιήσαμε.

```
%%time
ann_input = theano.shared(X_train)
ann_output = theano.shared(Y_train)
n_hidden=5
with pm.Model() as boston_nn_ARD:
    # Initialize random weights between each layer
    init_1 = np.random.randn(X_train.shape[1]*n_hidden)
    init_a = np.random.randn(n_hidden)
    init_b = np.random.randn()
    init_out = np.random.randn(n_hidden)
    #first lvl prior
    tau0=pm.Gamma('tau0',beta=0.001,alpha=0.1)
    #second lvl prior
    tau=pm.Gamma('tau',beta=0.1 / tau0,alpha=0.1,shape=(X_train.shape[1]))
    h=T.tile(tau,n_hidden)
    hh=T.nlinalg.diag(h)
    # ARD Prior for input to hidden layer
    weights_in_1 = pm.MvNormal('w_in_1',mu=T.zeros(X_train.shape[1]*n_hidden),
                               tau=hh,testval=init_1,shape=X_train.shape[1]*n_hidden)
    constants_in_1 = pm.Normal('c_in_1', 0, sd=1,shape=(n_hidden),testval=init_a)
    #hidden to output layer
    weights_1_out = pm.Normal('w_1_out', 0, sd=1,shape=(n_hidden),testval=init_out)
    constants_in_2 = pm.Normal('c_1_out', 0,sd=1,shape=(),testval=init_b)
    #sigma
    sigma = pm.HalfNormal('sigma', sd=1)
    #Feed-Forward Neural Network with relu activation
    weights_in_1=weights_in_1.reshape((13,5))
    act_1 = T.nnet.relu(T.dot(ann_input,weights_in_1) + constants_in_1)
    act_out = T.dot(act_1,weights_1_out) + constants_in_2
    # Likelihood (sampling distribution) of observations
    Y_obs = Normal('Y_obs', mu=act_out, sd=sigma, observed=ann_output)
```

Κώδικας 3.10: Ο κώδικας για το νευρωνικό δίκτυο με ARD πρότερη.

```
plt.plot(-inference_ARD.hist,'r',label='fullADVI', alpha=.3)
plt.legend()
plt.ylabel('ELBO')
plt.xlabel('iteration');
```




Γράφημα 3.11: Οι τιμές του ELBO ανά επανάληψη για το μοντέλο ARD.

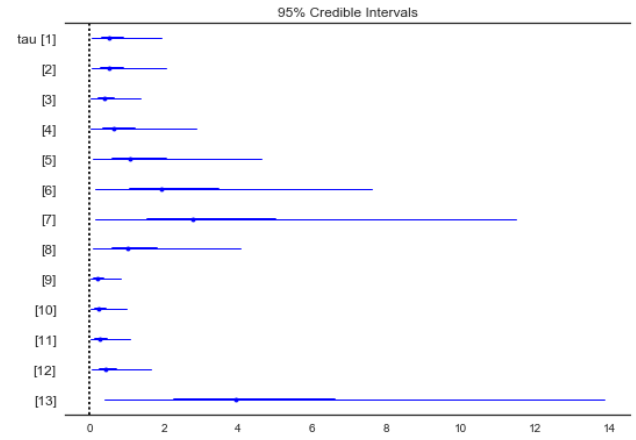
Στην συνέχεια προσομοιώσαμε 500 τιμές από την ύστερη προβλεπτική κατανομή για το σύνολο δεδομένων ελέγχου. Όσο αναφορά την προβλεπτική του ικανότητα του ARD, θα λέγαμε ότι είναι αρκετά ικανοποιητική και σίγουρα καλύτερη από το γραμμικό μοντέλο.

```
pred,pred2 = prediction_accuracy(tracefull_ARD,boston_nn_ARD,500)
100%|██████████████████████████████████| 500/500 [00:00<00:00, 911.93it/s]
Test MSE = [ 8.77817688]
Test MAE = [ 2.31084626]
Train NLPD = 2.626
```

Κώδικας 3.12: Η προβλεπτική ικανότητα του ARD μοντέλου.

Βέβαια όπως φαίνεται από τα 95% διαστήματα αξιοπιστίας της παραμέτρου $\tau_i$ (Γράφημα 3.13 ), δεν καταφέραμε να αναγνωρίσουμε κάποιες μεταβλητές ως μη σημαντικές, αφού καμία δεν είχε αρκετά μεγάλη ακρίβεια, δηλαδή μικρή διασπορά.

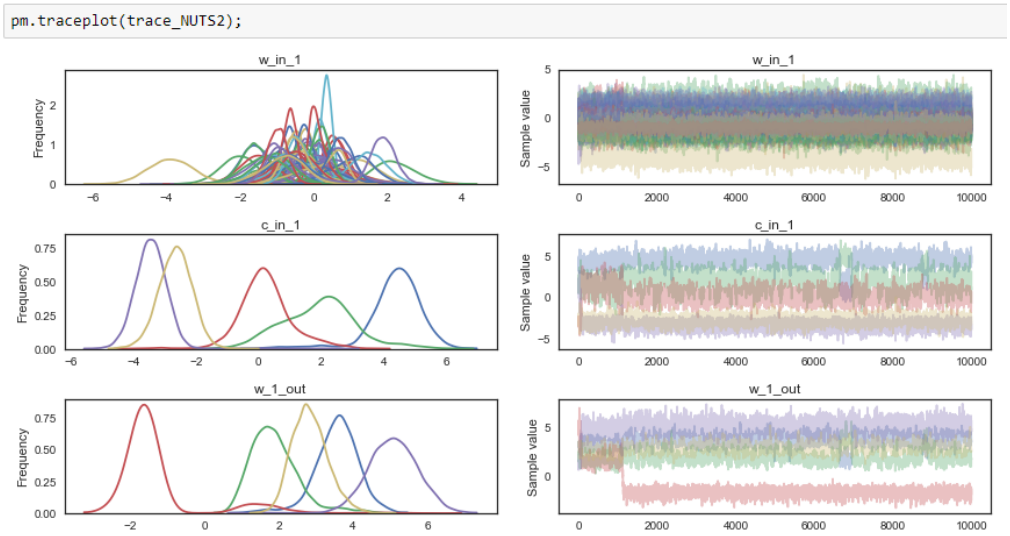
```
pm.forestplot(tracefull_ARD,varnames=['tau']);
```



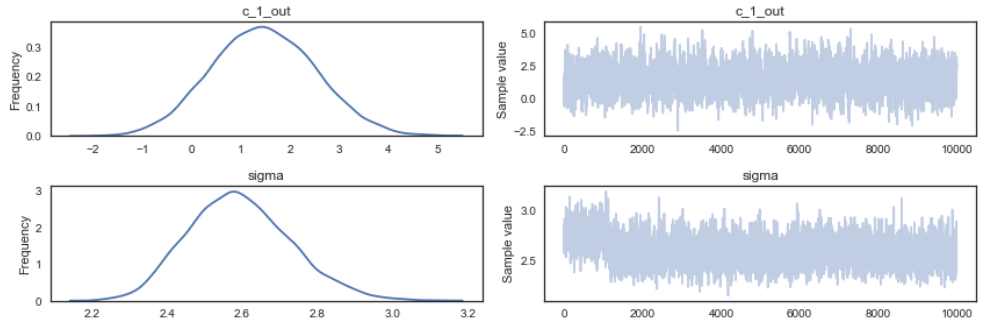

Γράφημα 3.13: Τα 95% διαστήματα αξιοπιστίας της παραμέτρου $\tau_i$.

Τα υπόλοιπα νευρωνικά δίκτυα που χρησιμοποιήσαμε είχαν αρκετά παρόμοια αποτελέσματα με το ARD. Αυτά που ξεχώρισαν ήταν το ARD και το μοντέλο με την τυποποιημένη Κανονική πρότερη. Στην συνέχεια διαλέξαμε το τελευταίο μοντέλο επειδή είχε τις λιγότερες παραμέτρους από τα υπόλοιπα, και χρησιμοποιήσαμε τον αλγόριθμο NUTS για τον υπολογισμό της ύστερης. Στην αρχή προσομοιώσαμε 2000 τιμές, αλλά επειδή η σύγκλιση δεν είχε επιτευχθεί για

κάποιες παραμέτρους, αποφασίσαμε να αυξήσουμε τις τιμές που θα προσομοιώσουμε σε 10000 τιμές. Τελικά καταφέραμε όχι μόνο να συγκλίνει η αλυσίδα, αλλά να έχει και καλύτερη προβλεπτική ικανότητα από τα προηγούμενα μοντέλα. Οι ύστερες κατανομές των παραμέτρων μας φαίνονται στο Γράφημα 3.14 και 3.15.




Γράφημα 3.14: Τα βάρη $\boldsymbol{\gamma}$ και $\beta_1, \ldots, \beta_H$ (NUTS).

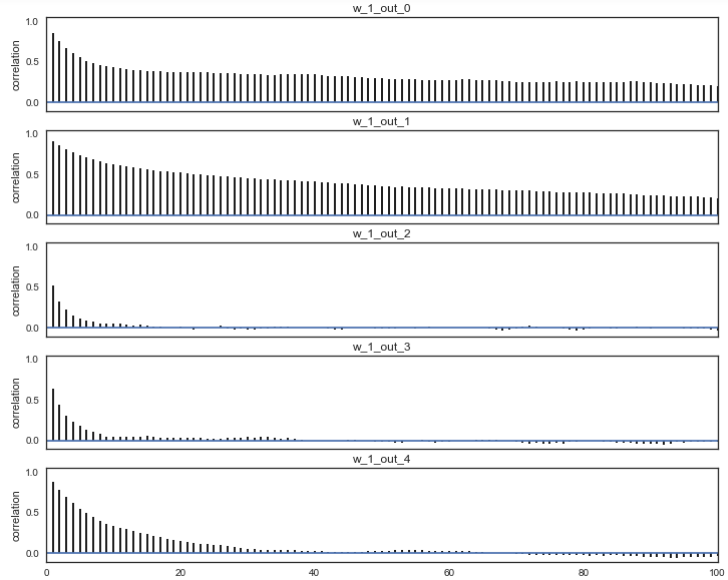



Γράφημα 3.15: Οι παράμετροι $\beta_0$ και $\sigma$(NUTS).

Το γράφημα για τα βάρη $\gamma_1, \gamma_2, \ldots$ δεν βοηθάει και πάρα πολύ, αφού βλέπουμε στο σύνολο $13 \times 5 = 65$ κατανομές σε ένα γράφημα. Από την άλλη η σταθερά $\gamma_0$ και τα βάρη $\beta$ φαίνονται ξεκάθαρα. Όπως παρατηρούμε από τα ιχνογράμματα ο αλγόριθμός μας είχε κάποιο πρόβλημα μέχρι και τις πρώτες 2000 τιμές και για αυτό η αλυσίδα είχε πρόβλημα στο να συγκλίνει. Επίσης η διασπορά $\sigma^2$ έχει

μικρότερη μέση τιμή σε σχέση με τα μοντέλα με τον ADVI που ήταν περίπου στο 3.6. Αυτό σίγουρα επηρεάζει την τιμή του NLPD, αλλά όχι τις υπόλοιπες δύο μετρικές. Σημαντικό είναι να αναφέρουμε ότι ο ADVI, αλλά και ο NUTS αυτή τη φορά δίνουν διαφορετικές κατανομές για τις παραμέτρους του μοντέλου, κάθε φορά που τους τρέχουμε για διαφορετικές αρχικές συνθήκες, όμως η προβλεπτική ικανότητα παραμένει ίδια. Όπως αναφέραμε και στο προηγούμενο κεφάλαιο, αυτή η διαφορά προκύπτει από το ότι πολλές παραμετρικοποιήσεις του ίδιου δικτύου είναι αποδεκτές.

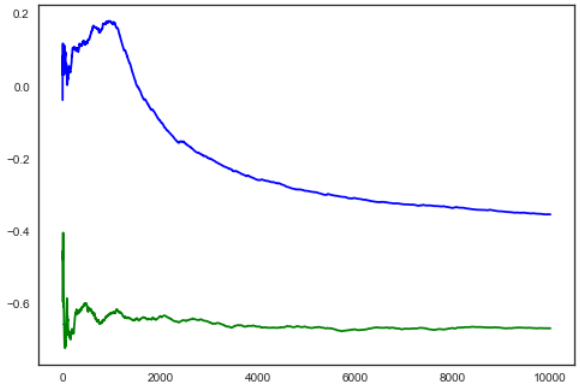
Λόγω του μεγάλου αριθμού των παραμέτρων, διαλέξαμε την μεταβλητή w_1_out (δηλαδή τα βάρη $\beta$ χωρίς τη σταθερά $\beta_0$ ) για να παρουσιάσουμε το διάγραμμα αυτοσυσχέτισης των τιμών της (Γράφημα 3.16). Ενώ για τον εργοδικό μέσο θα πάρουμε δύο παραμέτρους από τα βάρη $\gamma$.




Γράφημα 3.16: Το διάγραμμα αυτοσυσχέτισης των $\beta_1, \ldots, \beta_H$.

Στην πρώτη παράμετρο του διαγράμματος αυτοσυσχέτισης διακρίνουμε ένα πρόβλημα, το οποίο φαίνεται και από το ιχνόγραμμα της για τις πρώτες 2000 τιμές. Για αυτό το λόγω θα κρατήσουμε μόνο τις τελευταίες 8000 τιμές. Η σύγκλιση του εργοδικού μέσου έχει επιτευχθεί για όλες τις παραμέτρους του μοντέλου μας. Για λόγους απλότητας θα παρουσιάσουμε μόνο δυο από αυτές (Γράφημα 3.17).

```
R_M=ergodic_mean(trace_NUTS2.get_values(varname='w_in_1')[:,0,0])
R_M2=ergodic_mean(trace_NUTS2.get_values(varname='w_in_1')[:,1,4])
fig, ax1 = plt.subplots()
ax2 = ax1
ax1.plot(range(0,10000),R_M, 'g-');
ax2.plot(range(0,10000),R_M2, 'b-');
```




Γράφημα 3.17: Ο εργοδικός μέσος δυο παραμέτρων από τα βάρη $\gamma$.

Αφού είδαμε ότι χρησιμοποιώντας το MCMC μπορούμε να βελτιώσουμε την προβλεπτική ικανότητα του μοντέλου, χρησιμοποιήσαμε τον NUTS και για το Ιεραρχικό μοντέλο. Η σύγκλιση της αλυσίδας είχε τα ίδια προβλήματα με το μοντέλο με πρότερη την τυποποιημένη Κανονική, αλλά για άλλη μια φορά βελτιώθηκε η προβλεπτική του ικανότητα. Η διαφορά NUTS και του ADVI στο Ιεραρχικό μοντέλο φαίνονται παρακάτω (Κώδικας 3.18 και Κώδικας 3.19).

```
pred,pred2 = prediction_accuracy(tracefull_hm,boston_nn_hm,500)
```

```
100%|██████████████████████████████████| 500/500 [00:00<00:00, 1271.22it/s]
Test MSE = [ 8.78760488]
Test MAE = [ 2.31998821]
Train NLPD = 2.613
```

Κώδικας 3.18: Η προβλεπτική ικανότητα του Ιεραρχικού μοντέλου (ADVI)

Και τα δύο μοντέλα που χρησιμοποιήσαμε τον NUTS έχουν καλύτερη προβλεπτική ικανότητα από αυτά με τον ADVI. Γενικά τα μοντέλα με το ADVI δείχνουν πιο σταθερά, χωρίς κανένα πρόβλημα σύγκλισης και η προβλεπτική τους ικανότητα φαίνεται αρκετά καλή. Από την άλλη ο NUTS φαίνεται να έχει ένα πρόβλημα σύγκλισης, αλλά παρουσιάζει μικρότερο σφάλμα σε σχέση με το VI. Συνοψίζοντας, στην επόμενη σελίδα παρουσιάζουμε τα αποτελέσματα των μετρικών μας στο σύνολο ελέγχου Test_set για όλα τα μοντέλα που ανα-

```
pred,pred2 = prediction_accuracy(trace_NUTS_hm,boston_nn_hm,500)
100%|████████████████████████████████| 500/500 [00:00<00:00, 1208.37it/s]
Test MSE = [ 6.6464694]
Test MAE = [ 1.98500614]
Train NLPD = 2.541
```

Κώδικας 3.19: Η προβλεπτική ικανότητα του Ιεραρχικού μοντέλου (NUTS)

φέραμε (Πινακάκι 3.20). Στα μοντέλα όπου δεν αναφέρεται η πρότερη για την παράμετρο $\sigma$, χρησιμοποιήθηκε η περικομμένη στο 0 τυποποιημένη Κανονική.

| models | mse_test | mae_test | nlpd_test |
|---|---|---|---|
| Linear | 18.3 | 3.25 | 2.9 |
| NN flat prior (ADVI) | 8.37 | 2.28 | 2.63 |
| NN Normal(0,1) prior (ADVI) | 8.81 | 2.28 | 2.62 |
| NN Normal(0,1) prior (NUTS) | 6.54 | 1.98 | 2.48 |
| NN Normal(0,1) σ~flat (ADVI) | 8.07 | 2.27 | 2.59 |
| NN Normal(0,1) σ~Student (ADVI) | 8.3 | 2.33 | 2.6 |
| NN ARD prior (ADVI) | 8.78 | 2.31 | 2.63 |
| Hierarchical (ADVI) | 8.79 | 2.32 | 2.61 |
| Hierarchical (NUTS) | 6.64 | 1.99 | 2.54 |

Πινακάκι 3.20: Η προβλεπτική ικανότητα όλων των μοντέλων που χρησιμοποιήσαμε.

Το MCMC ανεξάρτητα από τα προβλήματα που αναφέραμε, είναι πιο ακριβές από την προσεγγιστική μέθοδο του ADVI. Βέβαια η διαφορά ως προς την ταχύτητα μεταξύ των δύο αλγορίθμων είναι τεράστια, αφού το ADVI χρειάζεται περίπου το $\frac{1}{4}$ του χρόνου που χρειάζεται ο NUTS να ολοκληρωθεί. Θα μπορούσαμε να πούμε ότι η προσεγγιστική μέθοδος ADVI δουλεύει καλύτερα όσο μεγαλώνει ο αριθμός των παραμέτρων του μοντέλου και στην περίπτωση ενός νευρωνικού δικτύου φαίνεται να διαχειρίζεται καλύτερα τις συσχετίσεις μεταξύ των παραμέτρων του. Κλείνοντας αυτήν την εργασία, θα ανεβάσουμε και άλλο τον αριθμό των παραμέτρων χρησιμοποιώντας ένα νευρωνικό δίκτυο με 2 κρυφά επίπεδα.

Το μοντέλο για την παλινδρόμηση είναι παρόμοιο, με τη διαφορά ότι αυτή τη φορά η συνάρτηση του κρυφού επιπέδου $h = h_m$ εκφράζει και τα δύο κρυφά επίπεδα:

$$
\begin{aligned}
&Y_i \sim N(f(\boldsymbol{x_i}), \sigma^2) \\
&\mathbb{E}[Y_i|\boldsymbol{x_i}] = f(\boldsymbol{x_i}) = o(\boldsymbol{x_i}) \\
&o(\boldsymbol{x_i}) = \beta_0 + \sum_{m=1}^{H} \beta_m \sigma(h_m(\boldsymbol{x_i})) \\
&h_m(\boldsymbol{x_i}) = \gamma'_0 + \sum_{j=1}^{H} \gamma'_{jm} \sigma(\boldsymbol{\gamma_{jm}}^T(\boldsymbol{x_i}) + \gamma_{j0m})
\end{aligned}
$$

Για να προσθέσουμε ένα ακόμα επίπεδο στο μοντέλο στο PyMC3 αρκεί να μετασχηματίσουμε λίγο των κώδικα. Η διαφορά είναι ότι έχουμε να ορίσουμε πρότερες και για τα βάρη $\boldsymbol{\gamma'}$ από το πρώτο κρυφό επίπεδο στο δεύτερο. Τώρα τα $\boldsymbol{\beta}$ είναι τα βάρη των νευρώνων από το δεύτερο κρυφό επίπεδο στο επίπεδο εξόδου και $\boldsymbol{\gamma}$ τα βάρη από το επίπεδο εισόδου στο πρώτο κρυφό επίπεδο. Σύνολο δημιουργήσαμε δύο μοντέλα, ένα με πρότερη την τυποποιημένη Κανονική κατανομή για όλες τις παραμέτρους $\boldsymbol{\gamma, \gamma', \beta}$ και ένα Ιεραρχικό πρώτης τάξης με Κανονική πρότερη με υπερπαράμετρο $\sigma_\star$ που ακολουθούσε Αντίστροφη Γάμμα. Για την διασπορά $\sigma$ η πρότερη που δώσαμε και στα δύο μοντέλα ήταν η περικομμένη στο μηδέν τυποποιημένη Κανονική. Ο κώδικας για το Ιεραρχικό μοντέλο στο φαίνεται παρακάτω (Κώδικας 3.21). Πρώτα τρέξαμε με τον ADVI το πρώτο μοντέλο για $4, 5, 6, 8$ νευρώνες σε κάθε κρυφό επίπεδο και στην συνέχεια δημιουργήσαμε ένα Ιεραρχικό μοντέλο με 4 νευρώνες σε κάθε κρυφό επίπεδο. Επιλέξαμε αυτό με τους 4, γιατί είχε την καλύτερη προβλεπτική ικανότητα από τα υπόλοιπα. Τέλος χρησιμοποιήσαμε και τον NUTS για τα δύο καλύτερα αυτά μοντέλα. Το Ιεραρχικό μοντέλο που είχε 3 παραμέτρους παραπάνω από το πρώτο, έφτασε τις 2 ώρες και 20 λεπτά για να ολοκληρωθεί με τον NUTS, ενώ με τον ADVI χρειάστηκε μόλις 10 λεπτά. Η προβλεπτική ικανότητα του ADVI σε αυτό το μοντέλο ήταν η καλύτερη που έχουμε δει μέχρι τώρα (Κώδικας 3.22).

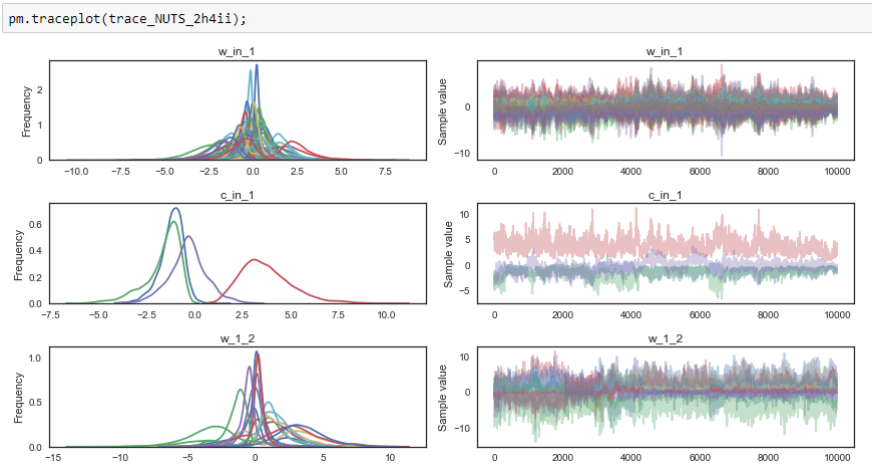
```
ann_input = theano.shared(X_train)
ann_output = theano.shared(Y_train)
n_hidden=4
with pm.Model() as boston_nn_2h4ii:
    # Initialize random weights between each layer
    init_1 = np.random.randn(X_train.shape[1], n_hidden)
    init_2 = np.random.randn(n_hidden, n_hidden)
    init_gamma = np.random.randn(n_hidden)
    init_gamma2 = np.random.randn(n_hidden)
    init_out = np.random.randn(n_hidden)
    init_beta = np.random.randn()
    #One lvl hyperparameter and Gaussian Priors for Input-to-Hidden layer
    sd1=pm.HalfNormal('sd1',sd=1)
    weights_in_1 = pm.Normal('w_in_1', 0, sd=sd1,
                             shape=(X_train.shape[1], n_hidden),testval=init_1)
    constant_in_1 = pm.Normal('c_in_1', 0, sd=sd1,shape=(n_hidden),testval=init_gamma)
    #One lvl hyperparameter and Gaussian Priors for Hidden-to-Hidden layer
    sd2=pm.HalfNormal('sd2',sd=1)
    weights_1_2 = pm.Normal('w_1_2', 0, sd=sd2,shape=(n_hidden, n_hidden),testval=init_2)
    constant_1_2 = pm.Normal('c_1_2', 0, sd=sd2,shape=(n_hidden),testval=init_gamma2)
    #One lvl hyperparameter and Gaussian Priors for Hidden-to-Output layer
    sd3=pm.HalfNormal('sd3',sd=1)
    weights_2_out = pm.Normal('w_2_out', 0, sd=sd3,shape=(n_hidden),
                                            testval=init_out)
    constant_2_out = pm.Normal('c_2_out', 0,sd=sd3, shape=(),testval=init_beta)
    #Set a Halfnormal prior for sd
    sigma = pm.HalfNormal('sigma', sd=1)
    #Feed-Forward Neural Network with relu activation
    act_1 = T.nnet.relu(T.dot(ann_input,weights_in_1)+constant_in_1)
    act_2 = T.nnet.relu(T.dot(act_1,weights_1_2)+constant_1_2)
    act_out = T.dot(act_2,weights_2_out)+constant_2_out
    #Set the likelihood of the model
    Y_obs = Normal('Y_obs', mu=act_out, sd=sigma, observed=ann_output)
```

Κώδικας 3.21: Ο κώδικας για το Ιεραρχικό μοντέλο με δυο κρυφά επίπεδα.

```
Average Loss = 1,039.9: 100%|████████| 200000/200000 [10:22<00:00, 321.20it/s]
Finished [100%]: Average Loss = 1,040.1
100%|████████████████████████████████| 500/500 [00:00<00:00, 1288.06it/s]

Test MSE = [ 7.31803899]
Test MAE = [ 2.15386221]
Train NLPD = 2.585
```

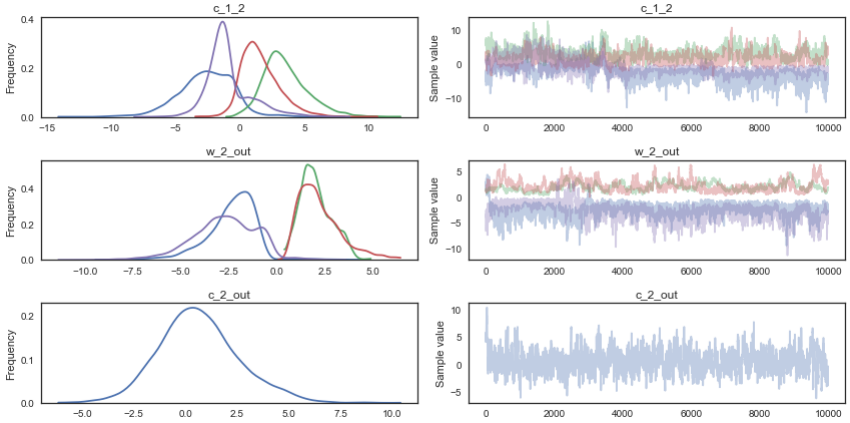
Κώδικας 3.22: Η προβλεπτική ικανότητα του Ιεραρχικού μοντέλου με δυο κρυφά επίπεδα (ADVI).

Χρησιμοποιώντας δύο κρυφά επίπεδα αυτή τη φορά, καταφέραμε να βελτιώσουμε λίγο την προβλεπτική ικανότητα του μοντέλου με τον ADVI. Όμως με μόλις ένα επίπεδο παραπάνω, τα προβλήματα του MCMC έγιναν ακόμα μεγαλύτερα, αφού προσθέσαμε στο μοντέλο και άλλες παραμέτρους. Τα συμπεράσματα και για τους δύο αλγορίθμους ήταν παρόμοια, ο NUTS είχε προβλήματα σύγκλισης και στα δύο μοντέλα που τον χρησιμοποιήσαμε. Συγκεκριμένα η υψηλή αυτοσυσχέτιση στο πρώτο μοντέλο μας έκανε να το απορρίψουμε και να δεχτούμε μόνο το Ιεραρχικό. Στα Γραφήματα 3.23,3.24,3.25 βλέπουμε τις ύστερες κατανομές των παραμέτρων μας και τα αντίστοιχα ιχνογράμματα που προσομειώθηκαν με το MCMC για το Ιεραρχικό μοντέλο.



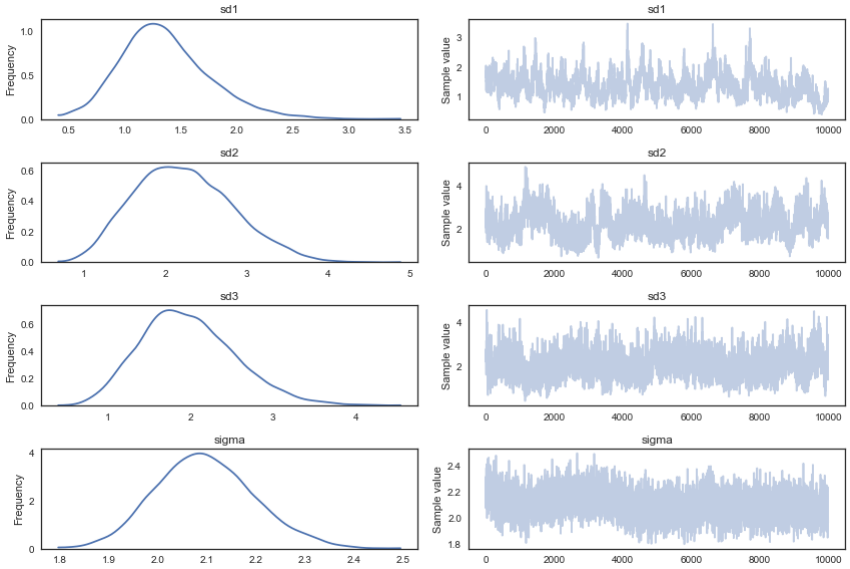

Γράφημα 3.23: Τα βάρη $\boldsymbol{\gamma}$ και $\boldsymbol{\gamma}'$, χωρίς της σταθερά $\gamma_0'$ (NUTS).

Η υψηλή αυτοσυσχέτιση φαίνεται αν κοιτάξουμε τα ινχογράμματα των παραμέτρων όλα μαζί. Υπάρχουν πολλά σημεία στα οποία η αλυσίδες διαφορετικών παραμέτρων ακολουθούν το ίδιο μοτίβο στην επιλογή των παραμέτρων. Από την άλλη οι διασπορές του μοντέλου τα πηγαίνουν λίγο καλύτερα (Γράφημα 3.25).



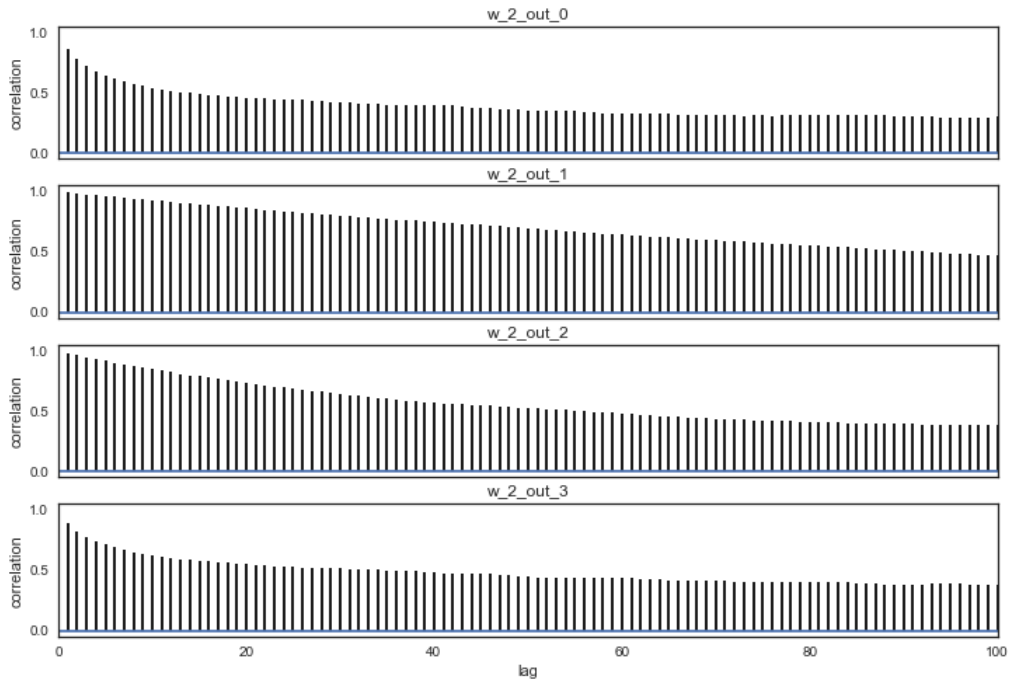

Γράφημα 3.24: Η σταθερά $\gamma_0'$ και τα βάρη $\boldsymbol{\beta}$(NUTS).

Για παράδειγμα μπορούμε να δούμε την αυτοσυσχέτιση των παραμέτρων $\beta_1, \ldots,$ $\beta_H = w\_2\_out$ (Γράφημα 3.26). Μάλλον θα πρέπει να προσομοιώσουμε παραπάνω τιμές και να περιμένουμε η αλυσίδα να συγκλίνει, βέβαια ίσως η αναμονή να μην αξίζει.




Γράφημα 3.25: Οι υπερπαράμετροι σε κάθε επίπεδο και η διασπορά $\sigma^2$(NUTS).

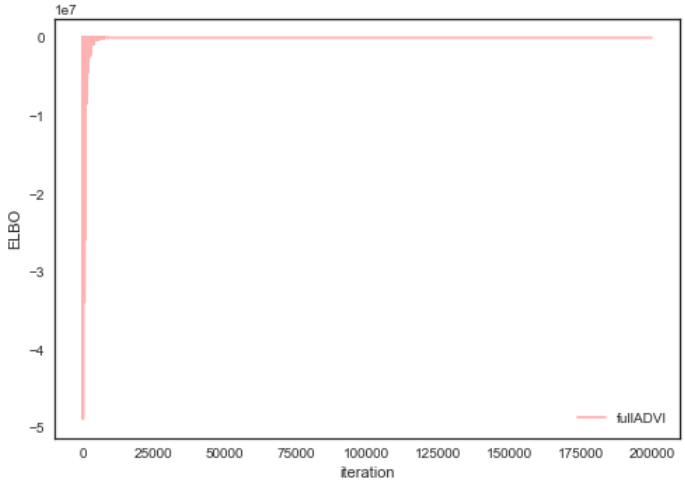
```
pm.autocorrplot(trace_NUTS_2h4ii[2000:],varnames=['w_2_out']);
```




Γράφημα 3.26: Το διάγραμμα αυτοσυσχέτισης για τα βάρη $\beta_1, \ldots, \beta_H$.

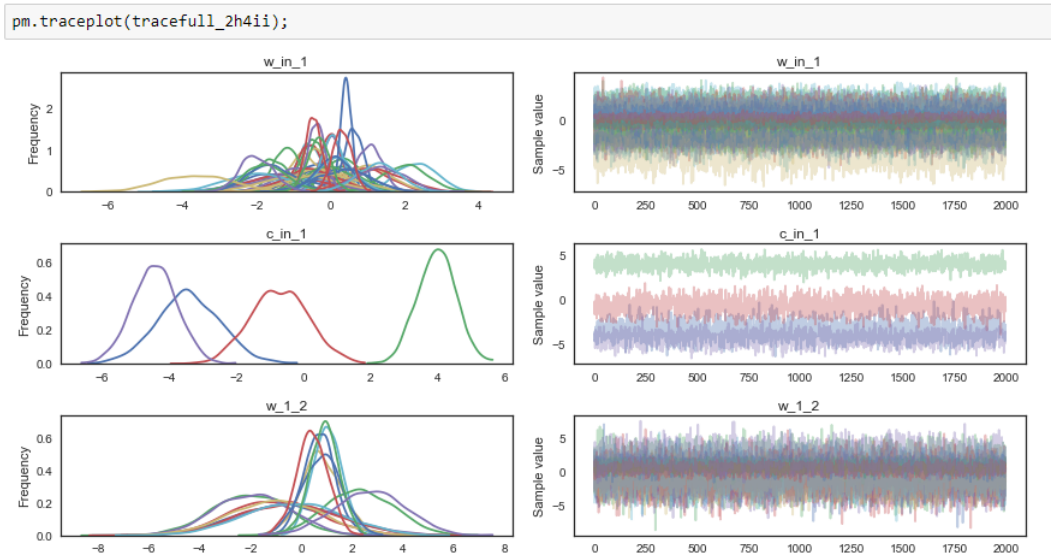
Για το ίδιο μοντέλο χρησιμοποιώντας τον ADVI, η σύγκλιση του $ELBO$ επετεύχθη αρκετά γρήγορα (Γράφημα 3.27).

```
plt.plot(-inference_2h4ii.hist,'r',label='fullADVI', alpha=.3)
plt.legend()
plt.ylabel('ELBO')
plt.xlabel('iteration');
```

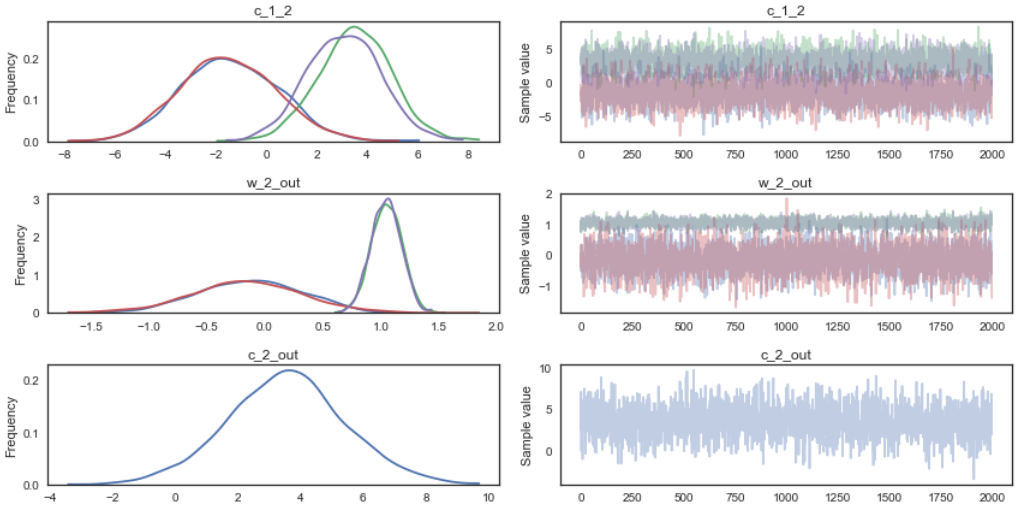



Γράφημα 3.27: Οι τιμές του ELBO ανά επανάληψη για το Ιεραρχικό μοντέλο με δυο κρυφά επίπεδα.

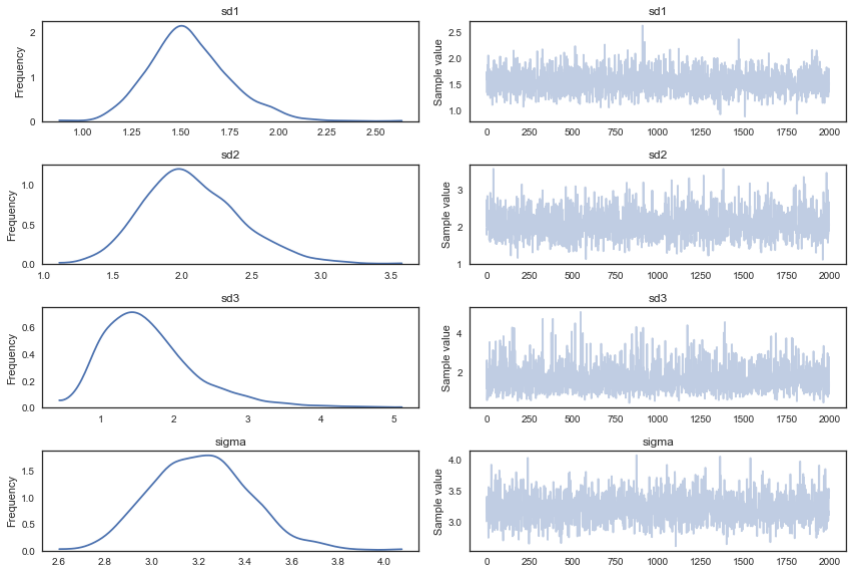
Ομοίως στο Γράφημα 3.28 και Γράφημα 3.29 βλέπουμε τις κατανομές των παραμέτρων και τα ιχνογραμματά τους. Το πρώτο επίπεδο έχει 52+4 παραμέτρους, ενώ το δεύτερο 16+4. Και εδώ παρατηρούμε μεγάλη ποικιλία στο σχήμα των κατανομών, ενώ υπάρχουν και κάποιες επίπεδες. Από τις υπερπαραμέτρους (Γράφημα 3.30) βλέπουμε πώς όσο περνάμε από το πρώτο επίπεδο στα επόμενα, τόσο μεγαλώνει η αβεβαιότητα των παραμέτρων μας. Η κατανομή της $sd1$ έχει μικρότερη μεταβλητότητα από τις επόμενες δύο, οι οποίες είναι αρκετά πιο επίπεδες. Τέλος δοκιμάσαμε και ένα Ιεραρχικό μοντέλο με 3 κρυφά επίπεδα και 4 νευρώνες σε κάθε επίπεδο, χρησιμοποιώντας μόνο τον ADVI. Τα αποτελέσματα ήταν χειρότερα από τα μοντέλα με το ένα κρυφό επίπεδο και άρα καταλήγουμε ότι δύο κρυφά επίπεδα είναι αρκετά για το πρόβλημά μας. Στο Πινακάκι 3.31 βλέπουμε την προβλεπτική ικανότητα των μοντέλων με πάνω από ένα κρυφά επίπεδα.




Γράφημα 3.28: Τα βάρη $\boldsymbol{\gamma}$ και $\boldsymbol{\gamma}'$, χωρίς της σταθερά $\gamma_0'$ (ADVI).




Γράφημα 3.29: Η σταθερά $\gamma_0'$ και τα βάρη $\boldsymbol{\beta}$ (ADVI).




Γράφημα 3.30: Οι υπερπαράμετροι σε κάθε επίπεδο και η διασπορά $\sigma^2$ (ADVI).

| | NN models | mse_test | mae_test | nlpd_test |
|---|---|---|---|---|
| **0** | Normal(0,1) 2 hidden 2 neurons(ADVI) | 8.51 | 2.30 | 2.69 |
| **1** | Normal(0,1) 2 hidden 4 neurons(ADVI) | 8.69 | 2.38 | 2.64 |
| **2** | Normal(0,σ~HNormal) 2 hidden 4 neurons(ADVI) | 7.30 | 2.15 | 2.59 |
| **3** | Normal(0,σ~HNormal) 2 hidden 4 neurons(NUTS) | 8.31 | 2.17 | 2.84 |
| **4** | NNormal(0,1) 2 hidden 5 neurons(ADVI) | 8.53 | 2.34 | 2.67 |
| **5** | Normal(0,1) 2 hidden 8 neurons(ADVI) | 9.02 | 2.34 | 2.84 |
| **6** | Normal(0,σ~HNormal) 3 hidden 4 neurons(ADVI) | 10.74 | 2.60 | 2.85 |

Πινακάκι 3.31: Η προβλεπτική ικανότητα των μοντέλων με πάνω από ένα κρυφά επίπεδα.

Κάνοντας μία σύνοψη των αποτελεσμάτων που είχαμε και για τα πραγματικά δεδομένα αυτού του κεφαλαίου, θα μπορούσαμε να πούμε ότι καταφέραμε να δημιουργήσουμε με επιτυχία ένα αρκετά καλό προβλεπτικό μοντέλο. Βέβαια, δοκιμάσαμε αρκετά μοντέλα και χωρίς την ταχύτητα του ADVI δεν θα τα είχαμε καταφέρει. Ο αλγόριθμός αυτός φαίνεται αρκετά υποσχόμενος για σύνθετα προβλήματα όπου ο αριθμός των παραμέτρων είναι αρκετά μεγάλος. Από την άλλη, ο NUTS είναι πιο ακριβείς, αλλά το μοντέλο χρειάζεται αρκετή προσοχή για να καταφέρουμε τις αλυσίδες να συγκλίνουν.

# Κεφάλαιο 4

# Παράρτημα

## 4.1 Python,Theano και PyMC3

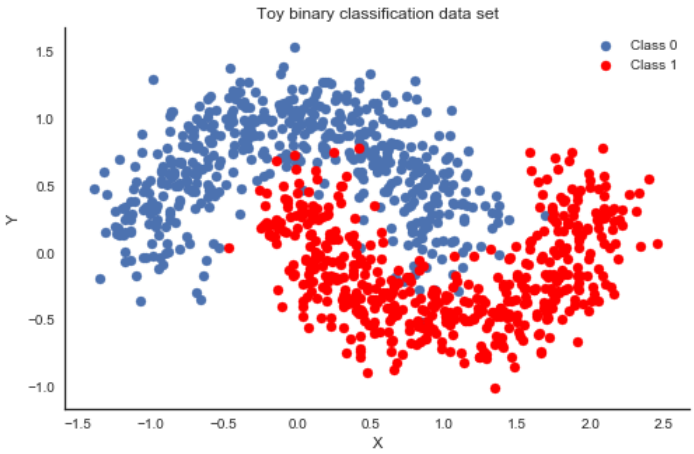
Η γλώσσα που επιλέξαμε για το προγραμματιστικό κομμάτι αυτής της εργασίας είναι η Python3.5. Στην συνέχεια θα περιγράψουμε λεπτομερώς τις βασικές εντολές που χρειάζεται κάποιος να γνωρίζει, ώστε να μπορεί να δημιουργήσει ένα Μπεϋζιανό μοντέλο παλινδρόμησης ή κατηγοριοποίησης χρησιμοποιώντας την βιβλιοθήκη PyMC3. Ξεκινάμε με ένα λογιστικό μοντέλο για το πιλοτικό πρόβλημα του κεφαλαίου 2. Πρώτα πρέπει να φορτώσουμε τις απαραίτητες βιβλιοθήκες στην Python. Η βασική εντολή για αυτό είναι είτε η import είτε η import as (αν θέλουμε να μετονομάσουμε την βιβλιοθήκη ώστε να μην κουβαλάμε το πλήρες όνομά της). Αν θέλουμε να φορτώσουμε μία συγκεκριμένη συνάρτηση ή πακέτο συναρτήσεων για να μην περιμένουμε να φορτώσουμε ολόκληρη τη βιβλιοθήκη, μπορούμε να χρησιμοποιήσουμε την εντολή from import as .

```
%matplotlib inline
import theano
import pymc3 as pm
import theano.tensor as T
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('white')
from sklearn import datasets
from sklearn.preprocessing import scale
from sklearn.cross_validation import train_test_split
from sklearn.datasets import make_moons
```

Κώδικας 4.1: Ο κώδικας για να φορτώσουμε τις βασικές βιβλιοθήκες της Python στο Jupyter Notebook.

Οι βασικές βιβλιοθήκες που θα χρησιμοποιήσουμε είναι η PyMC3 για Μπεϋζιανή Στατιστική και Πιθανοτικό Προγραμματισμό και η Theano για μαθηματικούς υπολογισμούς. Με την βιβλιοθήκη matplotlib.pyplot δημιουργούμε διάφορα γραφήματα και με την seaborn δίνουμε ένα λευκό μοτίβο στο υπόβαθρο.Τέλος θα χρησιμοποιήσουμε την βιβλιοθήκη numpy για μαθηματικούς υπολογισμούς που δεν έχει η Theano και την sklearn για διάφορες μετρικές, συναρτήσεις δειγματοληψίας και διάφορα άλλα στατιστικά εργαλεία που διαθέτει. Ας ξεκινήσουμε δημιουργώντας πρώτα συνθετικά δεδομένα για ένα πρόβλημα κατηγοριοποίησης ′0-1′

```
X, Y = make_moons(noise=0.2, random_state=0, n_samples=1000)
X = scale(X)
fig, ax = plt.subplots(figsize=(8, 5))
ax.scatter(X[Y==0, 0], X[Y==0, 1], label='Class 0')
ax.scatter(X[Y==1, 0], X[Y==1, 1], color='r', label='Class 1')
sns.despine(); ax.legend()
ax.set(xlabel='X', ylabel='Y', title='Toy binary classification data set');
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.5)
```




Γράφημα 4.2: Τα δεδομένα που δημιουργήσαμε με την συνάρτηση *make_moons*()

Δημιουργούμε 1000 ζευγάρια $(\boldsymbol{X_i}, Y_i)$, με $Y_i \in \{0, 1\}$ με την συνάρτηση make_moons() και με την scale() κανονικοποιούμε τα $\boldsymbol{X_i}$. Στην συνέχεια δημιουργούμε ένα γράφημα του δείγματός μας με την subplots(). Αυτή η συνάρτηση επιστρέφει δύο αντικείμενα, στην συγκεκριμένη περίπτωση την φιγούρα fig και τους άξονες ax. Πρώτα δημιουργούμε ένα διάγραμμα διασποράς για τα $\boldsymbol{X_i}$ που ανήκουν στην κλάση ′0′ με μπλε χρώμα, χρησιμοποιώντας την προέκταση .scatter() πάνω στην μεταβλητή ax. Στην συνέχεια κάνουμε το ίδιο και για την κλάση ′1′. Ουσιαστικά δημιουργήσαμε δύο υπογραφήματα το ένα πάνω στο άλλο. Με την προέκταση ax.set() δίνουμε ονόματα στους άξονες και τίτλο στο γράφημα και με την ax.legend() δημιουργούμε τιν λεζάντα για τις δύο διαφορε-

τικές κλάσεις στο προκαθορισμένο σημείο πάνω δεξιά. Τέλος με την συνάρτηση train_test_split() χωρίζουμε στη μέση τα δεδομένα μας δημιουργώντας δύο ξένα υποσύνολα δεδομένων το ένα για εκπαίδευση (προσαρμογή) του μοντέλου και το άλλο για πρόβλεψη. Γενικά στην Python οι προεκτάσεις που αναφέραμε είναι συναρτήσεις οι οποίες δρουν στο αντικείμενο πριν την τελεία π.χ. αν είχαμε ένα διάνυσμα (μεταβλητή) $\boldsymbol{x}$ και θέλαμε να πάρουμε την μέση τιμή του θα γράφαμε $x.mean()$, αν βέβαια ο τύπος της μεταβλητής υποστηρίζει αυτήν την συνάρτηση-προέκταση. Συνεχίζουμε δημιουργώντας το Μπεϋζιανό λογιστικό μοντέλο στο PyMC3

```
ann_input = theano.shared(X_train)
ann_output = theano.shared(Y_train)
with pm.Model() as logistic_regression:
    # Initialize random weights
    init_1 = np.random.randn(X_train.shape[1])
    init_b0 = np.random.randn()
    # Weights
    b = pm.Flat('b',shape=(X_train.shape[1]),
                                testval=init_1)
    b0 = pm.Flat('b0',shape=(),testval=init_b0)
    #Build the sigmoid function
    act_out = pm.math.sigmoid(pm.math.dot(ann_input,b) + b0)
    # Binary classification -> Bernoulli likelihood
    out = pm.Bernoulli('out',
                           act_out,
                           observed=ann_output,
                           total_size=Y_train.shape[0]
                          )
```

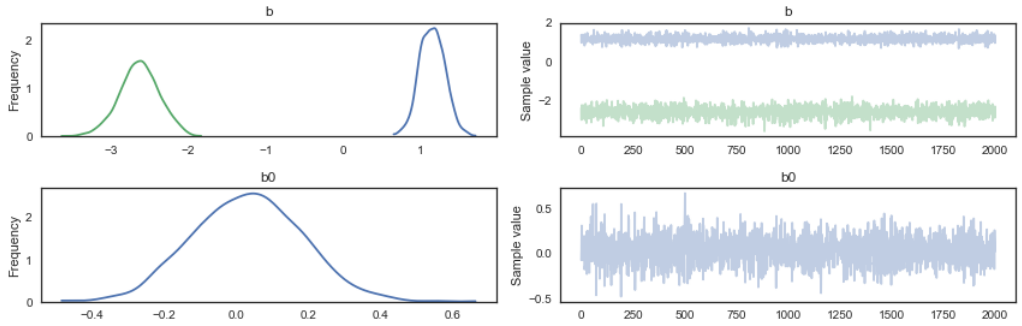
Κώδικας 4.3: Το λογιστικο μοντέλο παλινδρόμησης στο PyMC3.

Στην αρχή ορίζουμε δύο μεταβλητές ως theano.shared() μεταβλητές ώστε να έχουμε την δυνατότητα να αλλάξουμε την τιμή τους χωρίς να χρειάζεται να ξαναγράψουμε το μοντέλο. Με την εντολή with έχουμε την δυνατότητα να ανοίγουμε διάφορα αρχεία στην Python, τα οποία κλείνουν μετά την εκτέλεσή τους. Έτσι ανοίγουμε ένα μοντέλο στο PyMC3 με την εντολή pm.Model() δίνοντάς του το όνομα logistic regression. Ορίζουμε πρώτα δύο μεταβλητές init_1,init_b0 χρησιμοποιώντας την συνάρτηση του np.random.randn(size=size) η οποία παράγει τιμές από την τυποποιημένη κανονική κατανομή διάστασης size. Στην συνέχεια δίνουμε πρότερες κατανομές στους συντελεστές του μοντέλου $\beta_0, \beta = (\beta_1, \beta_2)$. Οι κατανομές στο PyMC3 είναι συναρτήσεις με ορίσματα ένα όνομα, τις παραμέτρους της κάθε κατανομές (στην συγκεκριμένη περίπτωση η flat δεν έχει), την διάσταση τους και μία αρχική τιμή για να τρέξει το πρώτο βήμα του αλγορίθμου. Η μεταβλητή act_out είναι η πιθανότητα $p$ του μοντέλου μας, η οποία χρειάζεται για να ορίσουμε την συνάρτηση πιθανοφάνειας του μοντέλου. Η πιθανότητα $p$ μοντελοποιείται μέσω της λογιστικής συνάρτησης, η οποία υπάρχει στην βιβλιοθήκη pm.math. Τέλος για να ορίσουμε την συνάρτηση

πιθανοφάνειας στο μοντέλο μας απλά δίνουμε την αντίστοιχη κατανομή στην μεταβλητή απόκρισης τους μοντέλου με το όνομα out. Για να υπολογίσουμε την ύστερη κατανομή θα χρησιμοποιήσουμε τον αλγόριθμο NUTS, προσομοιώνοντας 2000 τιμές. Ανοίγουμε το μοντέλο με την εντολή with και παράγουμε τιμές με την pm.sample(), η οποία χρησιμοποιεί ως προκαθορισμένο αλγόριθμο MCMC τον NUTS με αρχικές τιμές που δίνονται από τον αλγόριθμο ADVI. Με την συνάρτηση pm.traceplot() παίρνουμε ένα γράφημα της κατανομής των τιμών (Density Plot) καθώς και ένα ιχνόγραμμα (traceplot) των 2000 τιμών που προσομοιώσαμε:

```
with logistic_regression:
    trace_LR=pm.sample(2000)
pm.traceplot(trace_LR);
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using ADVI...
Average Loss = 142.4:   5%|▌          | 10791/200000 [00:06<01:49, 1734.58it/s]
Convergence archived at 10800
Interrupted at 10,800 [5%]: Average Loss = 217.57
100%|████████████████████████████████| 2500/2500 [00:09<00:00, 276.85it/s]
```




Γράφημα 4.4: Οι ύστερες κατανομές των παραμέτρων $\beta_0, \beta_1, \beta_2$ του λογιστικού μοντέλου παλινδρόμησης και τα αντίστοιχα ιχνογράμματα.

Στην συνέχεια χρησιμοποιούμε την συνάρτηση prediction_accuracy, την οποία εξηγήσαμε στο Κεφάλαιο 2, για να δούμε την προβλεπτική ικανότητα του μοντέλου.

```
pred = prediction_accuracy(trace_LR,logistic_regression,samples=500)
```

```
100%|████████████████████████████████| 500/500 [00:00<00:00, 1258.40it/s]
Test Accuracy = 85.6%
```

Κώδικας 4.5: Η προβλεπτική ικανότητα του λογιστικού μοντέλου παλινδρόμησης.

## 4.2 Το Θεώρημα του Cybenko(Συνέχεια)

**Λήμμα:** Κάθε συνεχής σιγμοειδής συνάρτηση είναι διακριτική.

**Απόδειξη:**
Υπενθυμίζουμε ότι μία σιγμοειδής συνάρτηση $\sigma$ ικανοποιεί την σχέση:

$$\sigma(x) = \begin{cases} 1 & x \to \infty \\ 0 & x \to -\infty. \end{cases} \tag{4.1}$$

Για κάθε $\boldsymbol{x}, \boldsymbol{\gamma_j} \in \mathbb{R}^m$ και $\gamma_{j0}, \lambda, \phi \in \mathbb{R}$ ορίζουμε:

$$\sigma_\lambda(\boldsymbol{x}) := \sigma(\lambda(\boldsymbol{\gamma_j^T x} + \gamma_{j0}) + \phi) = \begin{cases} \to 1 & \boldsymbol{\gamma_j^T x} + \gamma_{j0} > 0, \lambda \to \infty \\ \to 0 & \boldsymbol{\gamma_j^T x} + \gamma_{j0} < 0, \lambda \to \infty \\ = \sigma(\phi) & \boldsymbol{\gamma_j^T x} + \gamma_{j0} = 0, \forall\ \lambda. \end{cases} \tag{4.2}$$

Άρα όσο $\lambda \to \infty$, η $\sigma_\lambda(\boldsymbol{x}) \to g(\boldsymbol{x})$ με

$$g(\boldsymbol{x}) := \begin{cases} 1 & \boldsymbol{\gamma_j^T x} + \gamma_{j0} > 0 \\ 0 & \boldsymbol{\gamma_j^T x} + \gamma_{j0} < 0 \\ = \sigma(\phi) & \boldsymbol{\gamma_j^T x} + \gamma_{j0} = 0. \end{cases} \tag{4.3}$$

Έστω $\Pi_{\boldsymbol{\gamma_j}, \gamma_{j0}} := \{\boldsymbol{x} | \boldsymbol{\gamma_j^T x} + \gamma_{j0} = 0\}$ και $H_{\boldsymbol{\gamma_j}, \gamma_{j0}} := \{\boldsymbol{x} | \boldsymbol{\gamma_j^T x} + \gamma_{j0} > 0\}$. Επειδή η $|\sigma_\lambda(\boldsymbol{x})| \leqslant max(1, \sigma(\phi))$ για κάθε $\boldsymbol{x}$, μπορούμε να εφαρμόσουμε το Θεώρημα Κυριαρχημένης Σύγκλισης στην κατά σημείο σύγκλιση $\sigma_\lambda(\boldsymbol{x}) \to g(\boldsymbol{x})$:

$$\lim_{\lambda \to \infty} \int_{\mathbb{R}^m} \sigma_{\lambda(x)} d\mu(x) = \int_{\mathbb{R}^m} \lim_{\lambda \to \infty} \sigma_{\lambda(x)} d\mu(x) =$$
$$= \int_{\mathbb{R}^m} g(x) d\mu(x) = \sigma(\phi)\mu(\Pi_{\gamma_j, \gamma_0}) + \mu(H_{\gamma_j, \gamma_0})$$

για κάθε $\phi, \boldsymbol{\gamma_j}, \gamma_{j0}$. Άρα όταν $\int_{\mathbb{R}^m} \sigma(\boldsymbol{\gamma_j^T x} + \gamma_{j0}) d\mu(\boldsymbol{x}) = 0$ για κάθε $\boldsymbol{\gamma_j}, \boldsymbol{x}, \gamma_0$ το παραπάνω ολοκλήρωμα είναι μηδεν και άρα $\mu(H_{\boldsymbol{\gamma_j}, \gamma_{j0}}) = 0$ και $\sigma(\phi)\mu(\Pi_{\boldsymbol{\gamma_j}, \gamma_{j0}}) = 0$ για κάθε $\boldsymbol{\gamma_j}, \boldsymbol{x}, \gamma_{j0}, \phi$ αναγκαστικά, επειδή το μέτρο είναι μη αρνητική συνάρτηση. Επίσης, αφού το παραπάνω συμβαίνει για κάθε $\phi$ τότε $\mu(\Pi_{\boldsymbol{\gamma_j}, \gamma_{j0}}) = 0$. Αυτό που θέλομε να δείξουμε τώρα είναι ότι αν $\mu(H_{\boldsymbol{\gamma_j}, \gamma_{j0}}) = \mu(\Pi_{\boldsymbol{\gamma_j}, \gamma_{j0}}) = 0$ για κάθε $\boldsymbol{\gamma_j}$ και $\gamma_{j0}$, τότε αναγκαστικά συνεπάγεται ότι $\mu = 0$. Αν $h(\boldsymbol{x})$ η χαρακτηριστική συνάρτηση του συνόλου $[\gamma_{j0}, \infty)$ ορίζουμε

$$F(h) = \int_{\mathbb{R}^m} h(\boldsymbol{\gamma_j^T x}) d\mu(\boldsymbol{x})$$

Ο $F$ είναι φραγμένος στον $L^\infty(\mathbb{R}^m)$, αφού το $\mu$ είναι πεπερασμένο μέτρο. Αρκεί να παρατηρήσουμε ότι αν $\boldsymbol{\gamma_j^T x} \in [\gamma_{j0}, \infty)$, τότε $\boldsymbol{\gamma_j x} - \gamma_{j0} \geqslant 0$, και επειδή η $h$ είναι χαρακτηριστική συνάρτηση το παραπάνω ολοκλήρωμα γίνεται το άθροισμα των μέτρων δύο ξένων συνόλων $H_{\boldsymbol{\gamma_j}, -\gamma_{j0}}$ και $\Pi_{\boldsymbol{\gamma_j}, -\gamma_{j0}}$:

$$F(h) = \mu(H_{\boldsymbol{\gamma_j}, -\gamma_{j0}}) + \mu(\Pi_{\boldsymbol{\gamma_j}, -\gamma_{j0}}) = 0$$

Ομοίως αν όρίσουμε την $h(\boldsymbol{x})$ την χαρακτηριστική συνάρτηση του συνόλου $(\gamma_{j0}, \infty)$, τότε παλι έχουμε ότι $F(h) = 0$. Λόγω γραμμικότητας του $F$ έχουμε ότι αν $s(\boldsymbol{x})$ είναι συνάρτηση σκάλα ή απλή τότε:

$$F(s) = F(\sum_{i=0}^{\infty} c_i h(\boldsymbol{x})) = \sum_{i=0}^{\infty} c_i \int_{\mathbb{R}^m} h(\boldsymbol{\gamma_j^T x}) d\mu(\boldsymbol{x}) = 0$$

για κάποια $c_i \in \mathbb{R}$. Επειδή οι απλές συναρτήσεις είναι πυκνές στον $L^\infty(\mathbb{R}^m)$ συνεπάγεται ότι $F = 0$. Έστω τώρα δύο συναρτήσεις $y_1(\boldsymbol{x}) = sin(\boldsymbol{m^T x}), y_2(x) = cos(\boldsymbol{m^T x}) \in L^\infty(\mathbb{R}^m)$. Πάλι λόγω γραμμικότητας του $F$ έχουμε ότι

$$F(y_1 + iy_2) = \int_{\mathbb{R}^m} sin(\boldsymbol{m^T x}) + icos(\boldsymbol{m^T x}) d\mu(\boldsymbol{x}) = \int_{\mathbb{R}^m} e^{i\boldsymbol{m^T x}} d\mu(\boldsymbol{x}) = 0$$

Η παραπάνω σχέση μας δίνει ότι ο μετασχηματισμός Fourier του μέτρου $\mu$ είναι μηδέν, άρα θα πρέπει αναγκαστικά $\mu = 0$ και άρα ολοκληρώσαμε το τελευταίο μέρος της απόδειξης.

# Βιβλιογραφία


[1] http://twiecki.github.io/.

[2] http://www.ucl.ac.uk/~ucahad0/2010/3103_handout_3.pdf.

[3] http://www.ucl.ac.uk/~ucahad0/2010/3103_handout_6.pdf.

[4] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. *arXiv preprint arXiv:1611.01491*, 2016.

[5] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[6] Matthew D Hoffman and Andrew Gelman. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

[7] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *Journal of Machine Learning Research*, 18(14):1–45, 2017.

[8] Herbert KH Lee. *Bayesian nonparametrics via neural networks*. SIAM, 2004.

[9] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.