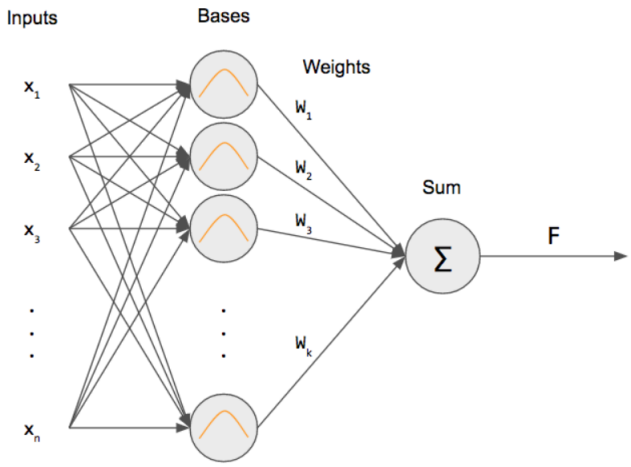
Επίλυση προβλήματος Παλινδρόμησης χρήση RBF νευρωνικού δικτύου
RBF Regression



Αριστοστέλειο Πανεπιστήμιο Θεσσαλονίκης
8ο Εξάμηνο
Τομέας Ηλεκτρονικής Υπολογιστών
Ασαφή Συστήματα - Υπολογιστική Νοημοσύνη

Χάρης Φίλης ΑΕΜ: 9449
Github Repository Link : here

October,2022

Contents

List of Figures

0.1 Abstract - Εισαγωγή

Στην παρούσα εργασία ανατίθεται η επίλυση του προβλήματος της παλινδρόμησης μέσω RBFNet. Συγκεκριμένα το δίκτυο που δημιουργώ έχει 2 κρυφά στρώματα εκ των οποίων το πρώτο εφαρμόζει το radial basis function kernel μετασχηματίζοντας τα δεδομένα εισόδου και στην συνέχεια προωθώντας τα σε ένα πυκνό layer το οποίο συσσωρεύει στην έξοδο τις τιμές προσπαθώντας να προσεγγίσει (παλινδρομικά) την συνάρτηση εκτίμησης τιμής ακινήτων στην Βοστόνη. Παρεμπιπτόντως το σύνολο δεδομένων που χρησιμοποιείται για την εκπαίδευση του νευρωνικού δικτύου είναι το boston housing dataset το οποίο φορτώνεται και εδώ μέσω βιβλιοθηκών της tensorflow. Ο λόγος που μελετάται RBFnet είναι η καλή υπό την γενική έννοια αποτελεσματικότητα των δικτύων αυτών ως λύτες προβλημάτων εκτίμησης συνάρτησης όπως στην συγκεκριμένη περίπτωση στο συγκεκριμένο παλινδρομικό μοντέλο, καθώς και λόγω του μικρού χρόνου εκπαίδευσης αυτών των δικτύων και της γρήγορης σύγκλισής τους όπως αναφέρεται και στην θεωρία. Στο πρώτο σκέλος της εργασίας δημιουργώ 3 διαφορετικά RBF δίκτυα με διαφορετικό αριθμό rbf νευρώνων τα οποία θα τα συγκρίνονται και στην συνέχεια ως προς την ικανότητα μοντελοποίησης αυτής της συνάρτησης κόστους των ακινήτων. Στο 2 ο σκέλος πραγματοποιείται fine-tuning ορισμένων υπερπαραμέτρων του παραπάνω με την μέθοδο grid-search και cross-validation όπως αυτή υλοποιήθηκε στο κομμάτι εργασίας του TSK. Το τελευταίο κομμάτι είναι αυτό που διαρκεί και περισσότερο στην εκτέλεση καθώς δημιουργούνται και εκπαιδεύονται συνολικά 48 δίκτυα με βάση του πλήθους διαφορετικών συνδυασμών των υπερπαραμέτρων που γίνονται tune.

0.2 Υλοποίηση **RBF** νευρωνικού δικτύου - **RBFnet Implementation**

Τα framework που χρησιμοποιήθηκαν για να στηθεί το νευρωνικό δίκτυο καθώς και να χειριστούν οι τανυστές του είναι τα Tensorflow και Keras. Παράλληλα για ορισμένες μετρικές καθώς και για το cross-validation split αλλά και για συναρτήσεις απόστασης χρησιμοποιήθηκαν οι βιβλιοθήκες sklearn και η scipy που έχουν καλές γραμμικές υλοποιήσεις συναρτήσεων χρήσιμων για το machine learning.
[1]

Στην δική μου υλοποίηση χρησιμοποίησα το Layer μοντέλο του Keras για στήσω ένα custom RBF Layer το οποίο μετασχηματίζει τα δεδομένα με τρόπο που είναι ίδιος με το να εφαρμόζεται ένα gaussian kernel στα δεδομένα. Σε αυτό το σημείο παρουσιαστηκαν αρχικά προβλήματα στο fit του keras στην πρώτη υλοποίηση και είχαν οδηγήσει σε πορεία ώστε να εφαρμόσω απλά rbf kernel στα δεδομένα μέσω της pairwise. Ωστόσο, διόρθωσα την υλοποίηση του RBFLayer και ειδικότερα τον τρόπο που εισάγεται σε ένα ακολουθιακό μοντέλο του Keras. Δεν έγινε χρήση της pairwise. Το RBF layer ορίζεται με βάση τον παρακάτω κώδικα:

```python
from tensorflow.keras.layers import Layer
from keras import backend as K
from keras.initializers import RandomNormal, Initializer, Constant
import numpy as np

class RBFLayer(Layer):
    """
        RBF Layer applying Gaussian Kernel
        # Arguments
        * units := number of neurons
        * initializer := weight or centroid initializer for each
        neuron Kmean is used
        * b := thresholds of output layer
    """
    def __init__(self,units,sigmaInit,initializer=None,b=1.0,**kwargs):
        self.init_betas = b
        self.units = units
        self.sigmaInit = sigmaInit
        if not initializer:
            self.initializer = RandomNormal(mean = 0)
        else:
            self.initializer = initializer
        super(RBFLayer, self).__init__(**kwargs)
    def build(self, input_shape):
        # I initialize the weights of rbf layer with kmeans
        self.W = self.add_weight(name='W',
                                 shape=(self.units,input_shape[1]),
                                 initializer=self.initializer,
                                 trainable=True)
        self.b  = self.add_weight(shape=(self.units,),
                                  initializer=Constant(value=self.init_betas),
                                  trainable=True)
        super(RBFLayer,self).build(input_shape)
    def call(self, inputs):
        diff = K.expand_dims(self.W)
        H = K.transpose(diff - K.transpose(inputs))
        # take L2 norm for the exponent of the radial basis function
        l2 = K.sum(K.pow(H,2), axis=1)
        res = K.exp((-self.b * l2)/(2*(self.sigmaInit**2)))
        return res

    def compute_output_shape(self, input_shape):
        return ( self.units,input_shape[0])

    # define get_config function to use model_from_jason
    def get_config(self):
        config = {
            'units': self.units
        }
        base_config = super(RBFLayer, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))
```

[1] Η tensorflow και το keras API κατ επέκταση δεν διαθέτουν stable RBF layer αλλά μόνο σε πειραματική μορφή έχουν το πακέτο *tf.keras.layers.experimental.RandomFourierFeatures* το οποίο δεν χρησιμοποιήθηκε στην περίπτωσή μας.

Ουσιαστικά τίθενται δυο variable προς training μια είναι τα W που αναφέρεται στα συναπτικά βάρη του rbf layer και γίνονται initialize μεσω το KMeans αλγορίθμου ο οποίο τρέχει συγκεκριμένα iterations(δεν περιμένουμε να συγκλίνει εδώ).Η δεύτερη είναι τα κατώφλια του στρώματος εξόδου και επηρεάζουν την έξοδο της radial basis activation function.

Στο κομμάτι της activation function κάθε νευρώνα που υλοποιείται στο κομμάτι call. Αυτό που υλοποιείται πρακτικά είναι ο μετασχηματισμός τον δεδομένων εφαρμόζοντας το kenel function:

$$f(x) = \frac{e^{-||x-c_i||^2}}{2 * \sigma_i^2} \tag{1}$$

Όπου c_i ειναι τα κέντρα των cluster αρχικοποίησης των βαρών W και στην συνέχεια αφαιρείται το κατώφλι b. Το sigma υπολογίζεται για κάθε μοντέλο ξεχωριστά μέσω της συνάρτησης *computeSigma(n_centers)*

$$\sigma = \frac{dmax}{\sqrt{2 * n\_centers}} \tag{2}$$

Όπου dmax ειναι η μέγιστη απόσταση μεταξύ των κέντρων των cluster και σε αυτό το σημείο γίνεται η χρήση της pdist μιας και δεν έχουμε δεδομένα λίγων διαστάσεων(συγκεκριμένα τα features των δεδομενων ειναι 13 και το target value είναι η τιμή του ακινήτου).

Στο κομμάτι του InitCentersKmeans γίνεται η χρήση του kmeans αλγορίθμου για να αρχικοποιήσει τα κέντρα των νευρώνων του RBF layer με βάση το training data και με μέγιστο αριθμό επαναλήψεων 100.

Τέλος για να ολοκληρωθεί το δίκτυο δημιουργώ ένα dense layer 128 νευρώνων με συνάρτηση ενεργοποίησης την relu για να μην υπάρχει πρόβλημα με τις τιμές των gradients το οποίο καταλήγει σε έναν νευρώνα ο οποίος δίνει την εκτίμηση του μοντέλου.

Τα βάρη του output layer αρχικοποιούνται με μια κανονική κατανομή με μηδενική μέση τιμή και μικρή διασπορά ώστε να μην έχουμε exploding gradients και γενικότερα να αποφύγουμε οποιαδήποτε αλλοίωση των μετασχηματισμένων δεδομένων από το RBF layer. Η εκπαίδευση του RBF δικτύου γίνεται με τον αλγόριθμο KMeans όπου επαναυπολογίζονται τα κέντρα των νευρώνων καθώς και η νέα διασπορά τους ενώ το υπόλοιπο MLP δίκτυο του keras με back-propagating gradients ή καλύτερα back-propagation.

Αυτός είναι ο λόγος που δεν μπορούμε να χρησιμοποιήσουμε ένα έτοιμο keras model και πρέπει να γίνει δική μας υλοποίηση του RBF layer.

Στην δεδομένη περίπτωση λέμε ότι έχουμε εκπαίδευση δύο σταδίων μία για το RBF layer και μία για το output layer.

0.3 Απλές εφαρμογές **RBFnet**

Στο πρώτο ζητούμενο της εργασίας ζητείται να γίνει εφαρμογή του παραπάνω μοντέλου με διαφορετικό αριθμό όμως rbf νευρώνων δηλαδή νευρώνων που εφαρμόζουν το kernel στα δεδομένα. Έτσι αλλάζει η μεταβλητή n_clusters ανά περίπτωση σε {0.1,0.5,0.9} του μεγέθους του training set. Επίσης να σημειωθεί ότι στην συγκεκριμένη εργασία το dataset είναι σχετικά μικρότερο και το split που γίνεται μεταξύ δεδομένων εκπαίδευσης και δεδομένων αξιολόγησης είναι 75% - 25%.

Η αλλαγή του n_clusters επηρεάζει άμεσα το W variable του rbf δικτύου και επομένως των νευρώνων του.

Ο optimizer που επιλέγεται ειναι ο Stochastic Gradient Descent (SGD) με learning rate lr = 0.001 και τρέχει για 100 εποχές. Εδώ να σημειώσω πως επειδή το δίκτυο ήταν σχετικά μικρό και εκπαιδευόταν γρήγορα δεκαπλασίασα τις εποχές εκπαίδευσης. Αυτό είχε ως αποτέλεσμα μια καλύτερη αποτύπωση learning/loss curve και των μετρικών για να μπορέσω να βγάλω καλύτερα συμπεράσματα.

Ως μετρικές του μοντέλου ορίζονται οι R^2 και η RMSE οι οποίες υλοποιούνται στον κώδικα και με την χρήση και του keras backend K.

Μετά από την εμπειρία με την εργασία στο MLP θεώρησα πως το minibatch training είναι ίσως μια καλή επιλογή για την εκπαίδευσης ενός δικτύου. Οπότε η εκπαίδευση γίνεται με batch_size = 32 δεδομένου ότι τα δεδομένα εκπαίδευσης περιέχουν συνολικά 379 δείγματα.

0.3.1 Μοντέλο **1**

Το πρώτο μοντέλο είχε πλήθος rbf νευρώνων ή πλήθος kernel στο μοντέλο ίσο με 10% του πλήθους των δεδομένων εκπαίδευσης δηλαδή 37 νευρώνες. Στο στρώμα εξόδου έχουμε 4864

νευρώνες οι οποίοι καταλήγουν σε έναν νευρώνα εκτίμησης της τιμής. Κατά την εκπαίδευση όλων των δικτύων της παρούσας εργασίας παρακρατείται και ένα 20% των δεδομένων εκπαίδευσης ως δεδομένα επικύρωσης. Παρακάτω φαίνονται τα γραφήματα της μετρικής RMSE καθώς και της αντικειμενικής συνάρτησης κόστους που είναι ίδιου τύπου καθώς και οι τιμές της μετρικής R^2.

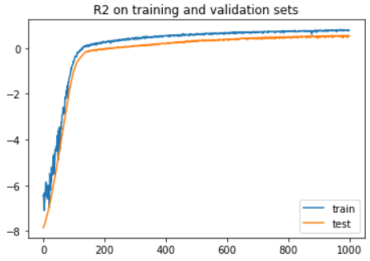


Figure 1: Model 1 R^2 metric slope

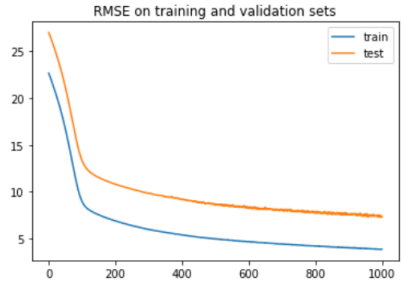


Figure 2: Model 1 $RMSE$ metric slope

Είναι φανερό πως το RMSE δεν καταλήγει άμεσα σε καλή τιμή ωστόσο μετά τις 100 εποχές έχουμε πολύ καλές τιμές και στις δύο εποχές τόσο για το validation όσο και για το training set. Επιπλέον, παρατηρείται ότι οι καμπύλες δεν μένουν στάσιμες και ακόμα και μέχρι την χιλιοστή εποχή η εκπαίδευση συνεχίζει να αποφέρει στην εκπαίδευση του δικτύου. Οι τελικές τιμές που φτάνει η εκπαίδευση μετά τις 1000 εποχές είναι:

- rmse = 4.167
- r2 = 0.74
- val_rmse = 7.39
- val_R2 = 0.51

Βλέπουμε σχετικά καλές τιμές και στο validation set και δεδομένου ότι η κλίση των καμπυλών έχει καλή πορεία θεωρείται ότι το μοντέλο γενικεύει καλά.

0.3.2 Μοντέλο 2

Το δεύτερο μοντέλο είχε πλήθος rbf νευρώνων ή πλήθος kernel στο μοντέλο ίσο με 50% του πλήθους των δειγμάτων του training set και όλα τα άλλα χαρακτηριστικά είναι ίδια με τον παραπάνω μοντέλο. Παρακάτω φαίνονται τα γραφήματα της μετρικής RMSE καθώς και της αντικειμενικής συνάρτησης κόστους που είναι ίδιου τύπου καθώς και οι τιμές της μετρικής R^2. Στο συγκεκριμένο

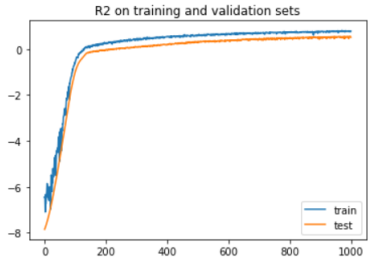


Figure 3: Model 2 R^2 metric slope

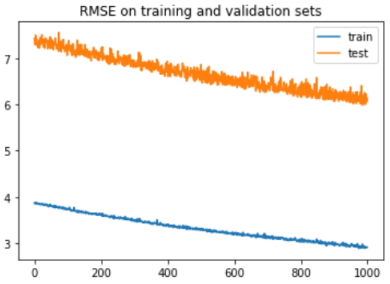


Figure 4: Model 2 $RMSE$ metric slope

γράφημα φαίνεται η παλινδρόμηση ακόμα και πριν τις 100 εποχές έχει λάβει καλή τιμή ωστόσο το RMSE test set φαίνεται να έχει μια σχετική απόσταση σε σχέση με αυτό του training set και αυτό δεδομένου ότι και τα δύο slopes είναι καθοδικά σημαινει underfit. Βέβαια αυτό δεν σημαίνει πως το μοντέλο δεν είναι καλό ωστόσο ίσως ο ιδανικός αριθμός cluster και δηλαδή rbf νευρώνων να είναι μεταξύ αυτής της περίπτωσης και της περίπτωσης

του πρώτου μοντέλου (37-120). Να σημειωθεί ότι το sigma = διασπορά των κέντρων των νευρώνων αναβαθμίζεται σε κάθε περίπτωση και τίθεται σε νέα τιμή με βάση την εξίσωση 2. Οι τελικές τιμές που φτάνει η εκπαίδευση μετά τις 1000 εποχές είναι:

- rmse = 3.15
- r2 = 0.87
- val_rmse = 6.0759
- val_R2 = 0.6338

Είναι σαφές λοιπόν πως το μοντέλο 2 είναι λίγο καλύτερο από το μοντέλο 1 ως προς τα τελικά score.

0.3.3 Μοντέλο 3

Το τρίτο μοντέλο είχε πλήθος rbf νευρώνων ή πλήθος kernel στο μοντέλο ίσο με 90% του πλήθους των δειγμάτων του training set και όλα τα άλλα χαρακτηριστικά είναι ίδια με τον παραπάνω μοντέλο.

Παρακάτω φαίνονται τα γραφήματα της μετρικής RMSE καθώς και της αντικειμενικής συνάρτησης κόστους που είναι ίδιου τύπου καθώς και οι τιμές της μετρικής R^2. Τελικές τιμές 1000 εποχής εκ-

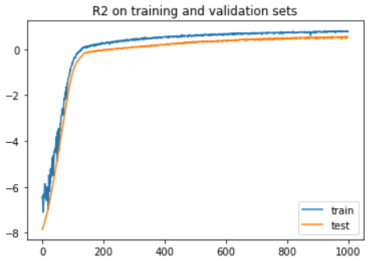


Figure 5: Model 3 R^2 metric slope

παίδευσης:

- rmse = 2.65
- r2 = 0.89
- val_rmse = 5.35
- val_R2 = 0.69

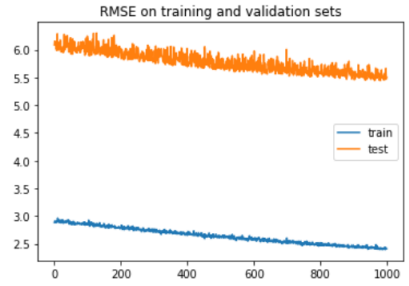


Figure 6: Model 3 $RMSE$ metric slope

Παρατηρούνται καλύτερα scores από όλα τα μοντέλα και πρακτικά είναι σαν να βρισκόμαστε πιο μετά στην εκπαίδευση προηγούμενων δικτύων. Οπότε φαίνεται σαν με κάποιο τρόπο αυξάνοντας τον αριθμό των πυρήνων ή μάλλον αυξάνοντας τον αριθμό των rbf νευρώνων να έχουμε το φαινόμενο καλύτερων αρχικών μετρικών και τελικών μετρικών. Αυτό βέβαια πάντα δεδομένου των αποτελεσμάτων που υπάρχουν και φαίνονται και ίσως προβληματίζει και η απόσταση του RMSE μεταξύ train και validation set στα δύο τελευταία μοντέλα για φαινόμενο underfitting.

0.3.4 Σύγκριση μοντέλων

Ως τελική κρίση φαίνεται το μοντέλο 1 εφόσον δεν εμφανίζει ούτε φαινόμενο overfitting ούτε underfitting να είναι το καλύτερο με αρκετά κοντά το μοντέλο 2 όπως είχε γίνει νωρίτερα η παρατήρηση.

0.4 Fine-Tuning RBFnet

0.4.1 Λίγα λόγια για το παραμετρικό μοντέλο

Στην συγκεκριμένη ενότητα μας ενδιαφέρει να ρυθμίσουμε ορισμένες υπερπαραμέτρους του RBF δικτύου το οποίο πλέον θα περιλαμβάνει και dropout layer πριν το output layer. Πιο συγκεκριμένα οι παράμετροι που θέτονται για βελτιστοποίηση είναι οι εξής:

- πλήθος rbf νευρώνων ή πυρήνων rbf n_rbf $\in \{5\%, 15\%, 30\%, 50\%\}$ του πλήθους των δειγμάτων εισόδου.
- πλήθος νευρώνων layer εξόδου n_hidden_2 $\in \{32, 64, 128, 256\}$

- πιθανότητα Dopout p $\in \{0.2, 0.35, 0.5\}$

Σε αυτό το κομμάτι αποφασίστηκε να εφαρμοστεί ο αλγόριθμος fine tuning παραμέτρων που χρησιμοποιήθηκε στην εργασία με τα TSK μοντέλα gridSearch με 5-fold cross validation split του dataset. Βάση των υπερπαραμέτρων του grid που είναι προς βελτιστοποίηση βλέπουμε ότι θα πρέπει να εκπαιδευτούν και αξιολογηθούν συνολικά 48 μοντέλα. Οι εποχές που γίνεται αυτό ορίζονται στις 100 και μάλιστα δημιουργείται σήμα Early Stopping με patience = 20 epochs για την αποφυγή άσκοπης εκπαίδευσής δικτύων. Μιας και τα RBF δίκτυα φάνηκε ήδη ότι εκπαιδεύονται γρήγορα χρησιμοποιήθηκε μικρό patience. Επίσης ο optimizer είναι και πάλι ο SGD με learning rate = 0.001 και γίνεται αρχικοποίηση των layers με την γνωστή ως τώρα κανονική κατανομή μηδενικής μέσης τιμής και μικρής διασποράς για να μην εμφανιστούν φαινόμενα exploding gradients και δούμε πολύ κακές τιμές της αντικειμενικής συνάρτησης κόστους.

Μετά από αρκετή ώρα εκτέλεσης του αλγορίθμου gridSearch (στο google Colab με χρήση gpu πήρε 1 ώρα περίπου) οι βέλτιστες τιμές των παραμέτρων που υπολογίστηκαν είναι οι εξής:

- optimal_rbf_neurons = 56
- optimal_n_hidden_2 = 256
- optimal_p = 0.35

Φαίνεται ότι ο βέλτιστος αριθμός rbf νευρώνων είναι ανάμεσα στο πρώτο και το δεύτερο μοντέλο ενώ θέλουμε τον μέγιστο αριθμό output layer νευρώνων για να γίνει καλύτερα συμψηφισμός του και να έχουμε καλύτερη παλινδρόμηση.

Layer (type)	Output Shape	Param #
rbf_layer_1 (RBFLayer)	(None, 56)	784
dense_2 (Dense)	(None, 256)	14592
dropout_1 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 1)	257

Total params: 15,633
Trainable params: 15,633
Non-trainable params: 0

Figure 7: Optimal Model Summary

0.4.2 Εκπαίδευση μοντέλου

Η εκπαίδευση του βέλτιστου μοντέλου έγινε στην βάση των 1000 εποχών καθώς είναι πιο εμφανή εκεί τα αποτελέσματα στα διαγράμματα. Παρατηρούμε

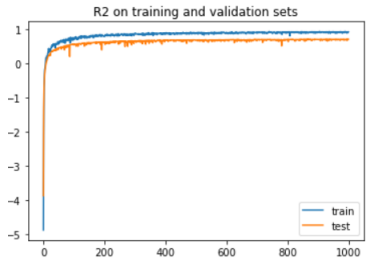


Figure 8: Optimal Model R^2 metric slope

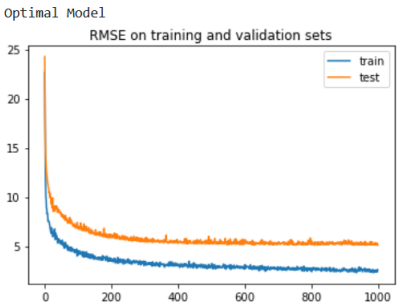


Figure 9: Optimal model $RMSE$ metric slope

ότι η μετρική παλινδρόμησης είναι αρκετά καλή $R^2 - > 1$ επίσης το RMSE τείνει στις πιο χαμηλές τιμές που έχουμε δει καθώς επίσης και οι αποστάσεις μεταξύ των test error και του train error είναι μικρή άρα μπορούμε να πούμε ότι το μοντέλο γενικεύει καλά.

0.4.3 Αξιολόγηση μοντέλου

Έγινε μια τυχαία αξιολόγηση του μοντέλου στο test set όπως φαίνεται παρακάτω χωρίς όμως εκπληκτικά αποτελέσματα.

```
# Evaluation Test
slice_length = 127
data_length = x_train.shape[0]
max_offset = data_length - slice_length
random_offset = tf.random.uniform((), minval=0, maxval=max_offset, dtype=tf.dtypes.int64)
slice_indices = tf.range(0, slice_length, dtype=tf.dtypes.int64)
random_slice = tf.gather(x_train, slice_indices + random_offset, axis=0)
metrics = optimal_model.evaluate(random_slice,y_test,verbose=1)

print('For the test set:')
print('MSE: {}'.format(metrics[0]))
print('RMSE: {}'.format(metrics[1]))
```

```
4/4 [==============================] - 0s 1ms/step - loss: 168.8268 - root_mean_squared_error: 12.9933
- R2: -1.2908
For the test set:
MSE: 168.8268280029297
RMSE: 12.993337631225586
```

Figure 10: Optimal Model Evaluation