ID A1812

Submission Date : 11/05/2019

House Prices: Advanced Regression Techniques

Abstract:

House Price prediction is a very popular dataset for data science competition. In this dataset 79 explanatory variables describing (almost) every aspect of residential homes in Ames and Iowa. This competition challenges competitor to predict the final price of each home.

In this report my main focus is how artificial neural network performs for this kind of problems and how to improve performance of the prediction using artificial neural network. So my elaboration on that section will be much more detailed.I have divided my work in four part and they are

- **Data processing** where I have visualized, cleaned, handled missing data, carefully modified , removed and merged some features.
- **Testing multiple model** In this part I have used gradient boosting, decision tree, random forest regression , lasso and Artificial neural network on my pre processed data.
- **Artificial neural network implementation** In this section I have implemented ANN , performed parameter tuning, training, used grid search inside training and validate test score.
- **Cross Validation** In this part I have used k fold cross validation on my artificial neural network model to make sure if the Data is actually independent and to fine tune few parameters on whole dataset if the cross validation score is not same as validation score.
- **Ensemble learning** I have used bagging method for this section to improve my kaggle score.

Score:

Best Score : 0.12192 (using Ensemble Learning)

output.csv 18 hours ago by Navid Using ['ANN_base_lr0.1_beta0.1-0.0-0.0-None_hidden16-8-4-None' 'ANN_lr0.1_beta0.1-0.05-0.0-0.0_hidden76-48-32-16' 'ANN_lr0.05_beta0.005-0.1-0.05-0.0_hidden8-32-16-8' 'ANN_lr0.05_beta0.1-0.0-0.0-0.0_hidden16-8-4-2' 'Random Forest Regressor' 'Xgboost' 'Lasso'] * [.15,.1,.1,.05,.0,.2,.4]	0.12192	☐

Best score without Ensemble : 0.12324 (ANN only)

output.csv a month ago by navid ann	0.12324	☑

Imports:

Gpu testing

In [4]:
```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  raise SystemError('GPU device not found')
print('Found GPU at: {}'.format(device_name))
```

```
Found GPU at: /device:GPU:0
```

In [5]:
```
import tensorflow as tf
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import  r2_score
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from IPython.display import Image
from sklearn.preprocessing import normalize,MinMaxScaler
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
# %matplotlib widget
%matplotlib inline
```

Data Pre-processing

Load Data

In [6]:
```
train = pd.read_csv('train.csv')#.select_dtypes(exclude=['object'])
test = pd.read_csv('test.csv')#.select_dtypes(exclude=['object'])

#look into datatypes of the file
print("data types count")
train.dtypes.groupby(train.dtypes).count()
```

```
data types count
```

Out[6]:
```
int64      35
float64     3
object     43
dtype: int64
```

Looking into data

In [7]:
```python
print('show sample')
pd.set_option('display.max_column', None)
train.head()
```

```
show sample
```

Out[7]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	H
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr	Norm	1Fam	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm	1Fam	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm	Norm	1Fam	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm	Norm	1Fam	

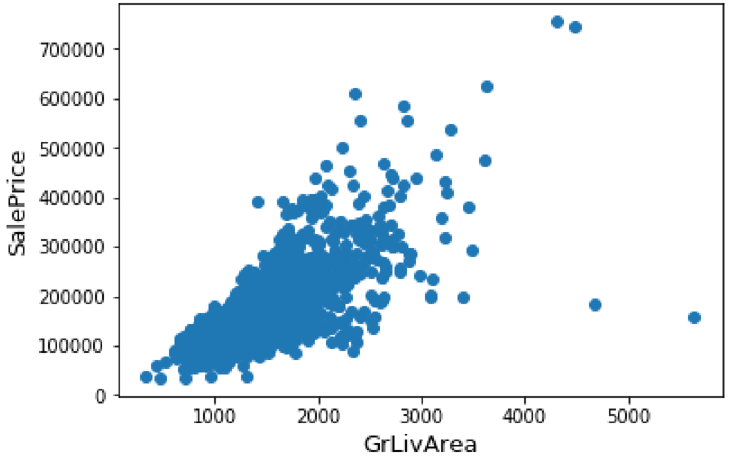
In [8]:
```python
print('description of data')
train.describe()
```

```
description of data
```

Out[8]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	Tot
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	1460.000000	1460.000000	14
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.639726	46.549315	567.240411	10
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.098091	161.319273	441.866955	4
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	0.000000	0.000000	
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.000000	0.000000	223.000000	7
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000	0.000000	477.500000	9
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.250000	0.000000	808.000000	12
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000	2336.000000	61

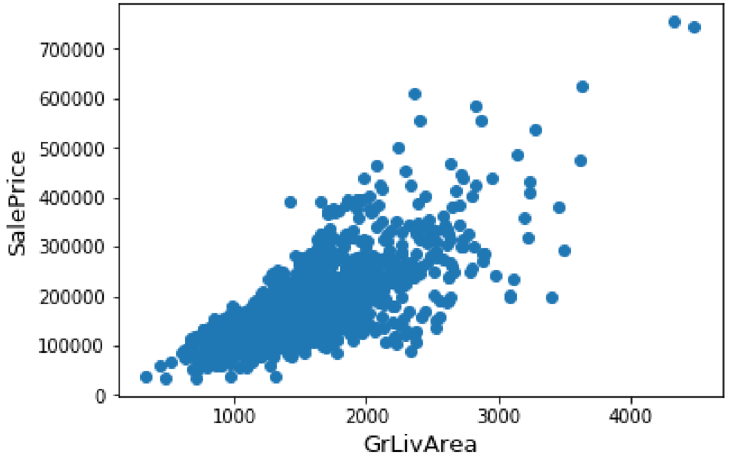
In [9]:
```python
fig, ax = plt.subplots()
ax.scatter(x = train['GrLivArea'], y = train['SalePrice'])
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)
plt.show()
```



There are a few houses with more than 4000 sq ft living area that are outliers, so we drop them from the training data.

In [10]:
```python
train.drop(train[ (train["GrLivArea"] > 4000) & (train['SalePrice']<400000) ].index, inplace=True)
```

In [11]:
```
#Check the graph again
fig, ax = plt.subplots()
ax.scatter(train['GrLivArea'], train['SalePrice'])
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)
plt.show()
```



RMSE

In [12]:
```
def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))
```

Imputing missing data

In [13]:
```
lot_frontage_by_neighborhood = train["LotFrontage"].groupby(train["Neighborhood"])
```

In [14]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

def factorize(df, factor_df, column, fill_na=None):
    factor_df[column] = df[column]
    if fill_na is not None:
        factor_df[column].fillna(fill_na, inplace=True)
    le.fit(factor_df[column].unique())
    factor_df[column] = le.transform(factor_df[column])
    return factor_df
```

common data processing:

In this part we have label encoded some of the columns because some features are ordinal. I have replaced some null value with zero because in those case they probably meant that it may not exist . Finally I have merged some of the features to get a better feature.

Befor starting following block its important to understand which feature means what so that describing my work would be easier

- SalePrice - the property's sale price in dollars. This is the target variable that you're trying to predict.
- MSSubClass: The building class
- MSZoning: The general zoning classification
- LotFrontage: Linear feet of street connected to property
- LotArea: Lot size in square feet
- Street: Type of road access
- Alley: Type of alley access
- LotShape: General shape of property
- LandContour: Flatness of the property
- Utilities: Type of utilities available
- LotConfig: Lot configuration
- LandSlope: Slope of property
- Neighborhood: Physical locations within Ames city limits
- Condition1: Proximity to main road or railroad
- Condition2: Proximity to main road or railroad (if a second is present)
- BldgType: Type of dwelling
- HouseStyle: Style of dwelling
- OverallQual: Overall material and finish quality
- OverallCond: Overall condition rating
- YearBuilt: Original construction date
- YearRemodAdd: Remodel date
- RoofStyle: Type of roof
- RoofMatl: Roof material
- Exterior1st: Exterior covering on house
- Exterior2nd: Exterior covering on house (if more than one material)
- MasVnrType: Masonry veneer type
- MasVnrArea: Masonry veneer area in square feet
- ExterQual: Exterior material quality
- ExterCond: Present condition of the material on the exterior
- Foundation: Type of foundation
- BsmtQual: Height of the basement
- BsmtCond: General condition of the basement
- BsmtExposure: Walkout or garden level basement walls
- BsmtFinType1: Quality of basement finished area
- BsmtFinSF1: Type 1 finished square feet
- BsmtFinType2: Quality of second finished area (if present)
- BsmtFinSF2: Type 2 finished square feet
- BsmtUnfSF: Unfinished square feet of basement area
- TotalBsmtSF: Total square feet of basement area
- Heating: Type of heating
- HeatingQC: Heating quality and condition

- CentralAir: Central air conditioning
- Electrical: Electrical system
- 1stFlrSF: First Floor square feet
- 2ndFlrSF: Second floor square feet
- LowQualFinSF: Low quality finished square feet (all floors)
- GrLivArea: Above grade (ground) living area square feet
- BsmtFullBath: Basement full bathrooms
- BsmtHalfBath: Basement half bathrooms
- FullBath: Full bathrooms above grade
- HalfBath: Half baths above grade
- Bedroom: Number of bedrooms above basement level
- Kitchen: Number of kitchens
- KitchenQual: Kitchen quality
- TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)
- Functional: Home functionality rating
- Fireplaces: Number of fireplaces
- FireplaceQu: Fireplace quality
- -GarageType: Garage location
- GarageYrBlt: Year garage was built
- GarageFinish: Interior finish of the garage
- GarageCars: Size of garage in car capacity
- GarageArea: Size of garage in square feet
- GarageQual: Garage quality
- GarageCond: Garage condition
- PavedDrive: Paved driveway
- WoodDeckSF: Wood deck area in square feet
- OpenPorchSF: Open porch area in square feet
- EnclosedPorch: Enclosed porch area in square feet
- 3SsnPorch: Three season porch area in square feet
- ScreenPorch: Screen porch area in square feet
- PoolArea: Pool area in square feet
- PoolQC: Pool quality
- Fence: Fence quality
- MiscFeature: Miscellaneous feature not covered in other categories
- MiscVal: $Value of miscellaneous feature
- MoSold: Month Sold
- YrSold: Year Sold
- SaleType: Type of sale
- SaleCondition: Condition of sale

In [15]:
```
def data_process(df):
    all_df = pd.DataFrame(index = df.index)

    all_df["LotFrontage"] = df["LotFrontage"]
    for key, group in lot_frontage_by_neighborhood:
        #Filling in missing LotFrontage values by the median
        idx = (df["Neighborhood"] == key) & (df["LotFrontage"].isnull())
        all_df.loc[idx, "LotFrontage"] = group.median()
        all_df["LotArea"] = df["LotArea"]

    all_df["MasVnrArea"] = df["MasVnrArea"]
    all_df["MasVnrArea"].fillna(0, inplace=True)

    all_df["BsmtFinSF1"] = df["BsmtFinSF1"]
    all_df["BsmtFinSF1"].fillna(0, inplace=True)

    all_df["BsmtFinSF2"] = df["BsmtFinSF2"]
    all_df["BsmtFinSF2"].fillna(0, inplace=True)

    all_df["BsmtUnfSF"] = df["BsmtUnfSF"]
    all_df["BsmtUnfSF"].fillna(0, inplace=True)

    all_df["TotalBsmtSF"] = df["TotalBsmtSF"]
    all_df["TotalBsmtSF"].fillna(0, inplace=True)

    all_df["1stFlrSF"] = df["1stFlrSF"]
    all_df["2ndFlrSF"] = df["2ndFlrSF"]
    all_df["GrLivArea"] = df["GrLivArea"]

    all_df["GarageArea"] = df["GarageArea"]
    all_df["GarageArea"].fillna(0, inplace=True)

    all_df["WoodDeckSF"] = df["WoodDeckSF"]
    all_df["OpenPorchSF"] = df["OpenPorchSF"]
    all_df["EnclosedPorch"] = df["EnclosedPorch"]
    all_df["3SsnPorch"] = df["3SsnPorch"]
    all_df["ScreenPorch"] = df["ScreenPorch"]

    all_df["BsmtFullBath"] = df["BsmtFullBath"]
    all_df["BsmtFullBath"].fillna(0, inplace=True)

    all_df["BsmtHalfBath"] = df["BsmtHalfBath"]
    all_df["BsmtHalfBath"].fillna(0, inplace=True)

    all_df["FullBath"] = df["FullBath"]
    all_df["HalfBath"] = df["HalfBath"]
    all_df["BedroomAbvGr"] = df["BedroomAbvGr"]
    all_df["KitchenAbvGr"] = df["KitchenAbvGr"]
    all_df["TotRmsAbvGrd"] = df["TotRmsAbvGrd"]
    all_df["Fireplaces"] = df["Fireplaces"]
```

```python
all_df["GarageCars"] = df["GarageCars"]
all_df["GarageCars"].fillna(0, inplace=True)

all_df["CentralAir"] = (df["CentralAir"] == "Y") * 1.0

all_df["OverallQual"] = df["OverallQual"]
all_df["OverallCond"] = df["OverallCond"]


"""following case are ordinal so we are performing label encoding here"""

nan = float('nan')
qual_dict = {nan: 0, "NA": 0, "Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5}
all_df["ExterQual"] = df["ExterQual"].map(qual_dict).astype(int)
all_df["ExterCond"] = df["ExterCond"].map(qual_dict).astype(int)
all_df["BsmtQual"] = df["BsmtQual"].map(qual_dict).astype(int)
all_df["BsmtCond"] = df["BsmtCond"].map(qual_dict).astype(int)
all_df["HeatingQC"] = df["HeatingQC"].map(qual_dict).astype(int)
all_df["KitchenQual"] = df["KitchenQual"].map(qual_dict).astype(int)
all_df["FireplaceQu"] = df["FireplaceQu"].map(qual_dict).astype(int)
all_df["GarageQual"] = df["GarageQual"].map(qual_dict).astype(int)
all_df["GarageCond"] = df["GarageCond"].map(qual_dict).astype(int)

all_df["BsmtExposure"] = df["BsmtExposure"].map(
    {nan: 0, "No": 1, "Mn": 2, "Av": 3, "Gd": 4}).astype(int)

bsmt_fin_dict = {nan: 0, "Unf": 1, "LwQ": 2, "Rec": 3, "BLQ": 4, "ALQ": 5, "GLQ": 6}
all_df["BsmtFinType1"] = df["BsmtFinType1"].map(bsmt_fin_dict).astype(int)
all_df["BsmtFinType2"] = df["BsmtFinType2"].map(bsmt_fin_dict).astype(int)

all_df["Functional"] = df["Functional"].map(
    {nan: 0, "Sal": 1, "Sev": 2, "Maj2": 3, "Maj1": 4,
     "Mod": 5, "Min2": 6, "Min1": 7, "Typ": 8}).astype(int)

all_df["GarageFinish"] = df["GarageFinish"].map(
    {nan: 0, "Unf": 1, "RFn": 2, "Fin": 3}).astype(int)

all_df["Fence"] = df["Fence"].map(
    {nan: 0, "MnWw": 1, "GdWo": 2, "MnPrv": 3, "GdPrv": 4}).astype(int)

all_df["PoolQC"] = df["PoolQC"].map(qual_dict).astype(int)

all_df["YearBuilt"] = df["YearBuilt"]
all_df["YearRemodAdd"] = df["YearRemodAdd"]

all_df["GarageYrBlt"] = df["GarageYrBlt"]
all_df["GarageYrBlt"].fillna(0.0, inplace=True)

all_df["MoSold"] = df["MoSold"]
all_df["YrSold"] = df["YrSold"]
```

```
all_df["LowQualFinSF"] = df["LowQualFinSF"]
all_df["MiscVal"] = df["MiscVal"]

all_df["PoolQC"] = df["PoolQC"].map(qual_dict).astype(int)

all_df["PoolArea"] = df["PoolArea"]
all_df["PoolArea"].fillna(0, inplace=True)

# Add categorical features as numbers too. It seems to help a bit.
all_df = factorize(df, all_df, "MSSubClass")
all_df = factorize(df, all_df, "MSZoning", "RL")
all_df = factorize(df, all_df, "LotConfig")
all_df = factorize(df, all_df, "Neighborhood")
all_df = factorize(df, all_df, "Condition1")
all_df = factorize(df, all_df, "BldgType")
all_df = factorize(df, all_df, "HouseStyle")
all_df = factorize(df, all_df, "RoofStyle")
all_df = factorize(df, all_df, "Exterior1st", "Other")
all_df = factorize(df, all_df, "Exterior2nd", "Other")
all_df = factorize(df, all_df, "MasVnrType", "None")
all_df = factorize(df, all_df, "Foundation")
all_df = factorize(df, all_df, "SaleType", "Oth")
all_df = factorize(df, all_df, "SaleCondition")

"""In following code I am converting values of those features as 0 or 1"""

# IR2 and IR3 don't appear that often, so just make a distinction
# between regular and irregular.
all_df["IsRegularLotShape"] = (df["LotShape"] == "Reg") * 1

# Most properties are level; bin the other possibilities together
# as "not level".
all_df["IsLandLevel"] = (df["LandContour"] == "Lvl") * 1

# Most land slopes are gentle; treat the others as "not gentle".
all_df["IsLandSlopeGentle"] = (df["LandSlope"] == "Gtl") * 1

# Most properties use standard circuit breakers.
all_df["IsElectricalSBrkr"] = (df["Electrical"] == "SBrkr") * 1

# About 2/3rd have an attached garage.
all_df["IsGarageDetached"] = (df["GarageType"] == "Detchd") * 1

# Most have a paved drive. Treat dirt/gravel and partial pavement
# as "not paved".
all_df["IsPavedDrive"] = (df["PavedDrive"] == "Y") * 1

# The only interesting "misc. feature" is the presence of a shed.
all_df["HasShed"] = (df["MiscFeature"] == "Shed") * 1.

# If YearRemodAdd != YearBuilt, then a remodeling took place at some point.
```

```python
    all_df["Remodeled"] = (all_df["YearRemodAdd"] != all_df["YearBuilt"]) * 1

    # Did a remodeling happen in the year the house was sold?
    all_df["RecentRemodel"] = (all_df["YearRemodAdd"] == all_df["YrSold"]) * 1

    # Was this house sold in the year it was built?
    all_df["VeryNewHouse"] = (all_df["YearBuilt"] == all_df["YrSold"]) * 1

    all_df["Has2ndFloor"] = (all_df["2ndFlrSF"] == 0) * 1
    all_df["HasMasVnr"] = (all_df["MasVnrArea"] == 0) * 1
    all_df["HasWoodDeck"] = (all_df["WoodDeckSF"] == 0) * 1
    all_df["HasOpenPorch"] = (all_df["OpenPorchSF"] == 0) * 1
    all_df["HasEnclosedPorch"] = (all_df["EnclosedPorch"] == 0) * 1
    all_df["Has3SsnPorch"] = (all_df["3SsnPorch"] == 0) * 1
    all_df["HasScreenPorch"] = (all_df["ScreenPorch"] == 0) * 1

    # Months with the largest number of deals may be significant.
#     mx = max(train["MoSold"].groupby(train["MoSold"]).count())
#     all_df["HighSeason"] = df["MoSold"].replace(
#         train["MoSold"].groupby(train["MoSold"]).count()/mx)

#     mx = max(train["MSSubClass"].groupby(train["MSSubClass"]).count())
#     all_df["NewerDwelling"] = df["MSSubClass"].replace(
#         train["MSSubClass"].groupby(train["MSSubClass"]).count()/mx)

    # following portion was calculated with above commented part of the code.
    # Instead of the fraction value putting binary value helps for generalization
    all_df["HighSeason"] = df["MoSold"].replace(
        {1: 0, 2: 0, 3: 0, 4: 1, 5: 1, 6: 1, 7: 1, 8: 0, 9: 0, 10: 0, 11: 0, 12: 0})

    all_df["NewerDwelling"] = df["MSSubClass"].replace(
        {20: 1, 30: 0, 40: 0, 45: 0,50: 0, 60: 1, 70: 0, 75: 0, 80: 0, 85: 0,
         90: 0, 120: 1, 150: 0, 160: 0, 180: 0, 190: 0})


    all_df.loc[df.Neighborhood == 'NridgHt', "Neighborhood_Good"] = 1
    all_df.loc[df.Neighborhood == 'Crawfor', "Neighborhood_Good"] = 1
    all_df.loc[df.Neighborhood == 'StoneBr', "Neighborhood_Good"] = 1
    all_df.loc[df.Neighborhood == 'Somerst', "Neighborhood_Good"] = 1
    all_df.loc[df.Neighborhood == 'NoRidge', "Neighborhood_Good"] = 1
    all_df["Neighborhood_Good"].fillna(0, inplace=True)

    # House completed before sale or not
    all_df["SaleCondition_PriceDown"] = df.SaleCondition.replace(
        {'Abnorml': 1, 'Alloca': 1, 'AdjLand': 1, 'Family': 1, 'Normal': 0, 'Partial': 0})

    # House completed before sale or not
    all_df["BoughtOffPlan"] = df.SaleCondition.replace(
        {"Abnorml" : 0, "Alloca" : 0, "AdjLand" : 0, "Family" : 0, "Normal" : 0, "Partial" : 1})

    all_df["BadHeating"] = df.HeatingQC.replace(
```

```
    {'Ex': 0, 'Gd': 0, 'TA': 0, 'Fa': 1, 'Po': 1})

area_cols = ['LotFrontage', 'LotArea', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
             'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'GarageArea', 'WoodDeckSF',
             'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'LowQualFinSF', 'PoolArea' ]
all_df["TotalArea"] = all_df[area_cols].sum(axis=1)

all_df["TotalArea1st2nd"] = all_df["1stFlrSF"] + all_df["2ndFlrSF"]

all_df["Age"] = 2010 - all_df["YearBuilt"]
all_df["TimeSinceSold"] = 2010 - all_df["YrSold"]

all_df["SeasonSold"] = all_df["MoSold"].map({12:0, 1:0, 2:0, 3:1, 4:1, 5:1,
                                              6:2, 7:2, 8:2, 9:3, 10:3, 11:3}).astype(int)

all_df["YearsSinceRemodel"] = all_df["YrSold"] - all_df["YearRemodAdd"]

# Simplifications of existing features into bad/average/good.
all_df["SimplOverallQual"] = all_df.OverallQual.replace(
    {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2, 6 : 2, 7 : 3, 8 : 3, 9 : 3, 10 : 3})
all_df["SimplOverallCond"] = all_df.OverallCond.replace(
    {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2, 6 : 2, 7 : 3, 8 : 3, 9 : 3, 10 : 3})
all_df["SimplPoolQC"] = all_df.PoolQC.replace(
    {1 : 1, 2 : 1, 3 : 2, 4 : 2})
all_df["SimplGarageCond"] = all_df.GarageCond.replace(
    {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
all_df["SimplGarageQual"] = all_df.GarageQual.replace(
    {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
all_df["SimplFireplaceQu"] = all_df.FireplaceQu.replace(
    {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
all_df["SimplFireplaceQu"] = all_df.FireplaceQu.replace(
    {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
all_df["SimplFunctional"] = all_df.Functional.replace(
    {1 : 1, 2 : 1, 3 : 2, 4 : 2, 5 : 3, 6 : 3, 7 : 3, 8 : 4})
all_df["SimplKitchenQual"] = all_df.KitchenQual.replace(
    {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
all_df["SimplHeatingQC"] = all_df.HeatingQC.replace(
    {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
all_df["SimplBsmtFinType1"] = all_df.BsmtFinType1.replace(
    {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2, 6 : 2})
all_df["SimplBsmtFinType2"] = all_df.BsmtFinType2.replace(
    {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2, 6 : 2})
all_df["SimplBsmtCond"] = all_df.BsmtCond.replace(
    {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
all_df["SimplBsmtQual"] = all_df.BsmtQual.replace(
    {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
all_df["SimplExterCond"] = all_df.ExterCond.replace(
    {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
all_df["SimplExterQual"] = all_df.ExterQual.replace(
    {1 : 1, 2 : 1, 3 : 1, 4 : 2, 5 : 2})
```

```python
    # Bin by neighborhood (a little arbitrarily). Values were computed by:
    # train_df["SalePrice"].groupby(train_df["Neighborhood"]).median().sort_values()
    neighborhood_map = {
        "MeadowV" : 0,  #  88000
        "IDOTRR" : 1,   # 103000
        "BrDale" : 1,   # 106000
        "OldTown" : 1,  # 119000
        "Edwards" : 1,  # 119500
        "BrkSide" : 1,  # 124300
        "Sawyer" : 1,   # 135000
        "Blueste" : 1,  # 137500
        "SWISU" : 2,    # 139500
        "NAmes" : 2,    # 140000
        "NPkVill" : 2,  # 146000
        "Mitchel" : 2,  # 153500
        "SawyerW" : 2,  # 179900
        "Gilbert" : 2,  # 181000
        "NWAmes" : 2,   # 182900
        "Blmngtn" : 2,  # 191000
        "CollgCr" : 2,  # 197200
        "ClearCr" : 3,  # 200250
        "Crawfor" : 3,  # 200624
        "Veenker" : 3,  # 218000
        "Somerst" : 3,  # 225500
        "Timber" : 3,   # 228475
        "StoneBr" : 4,  # 278000
        "NoRidge" : 4,  # 290000
        "NridgHt" : 4,  # 315000
    }

    all_df["NeighborhoodBin"] = df["Neighborhood"].map(neighborhood_map)
    return all_df
```

In the above block I have done following operations :

- Filled with 0 for some features like MasVnrArea , GarageArea , TotalBsmtSF etc .According to the documentation of the dataset if these features have any field empty then that means the feature is not available. So I have done this operation according to documentation of the dataset.
- CentralAir feature was given has two field only 'Y' or 'N' so I have converted that to 0 or 1
- For some ordinal features I ave performed lable encoding. Those features are ExterQual , ExterCond, BsmtQual, BsmtCond, HeatingQC, KitchenQual etc
- I have converted some features from categorical to numerical and those features are MSSubClass, MSZoning , LotConfig, RL , LotConfig,Neighborhood, Condition1 ,BldgType, HouseStyle , HouseStyle, Exterior1st, Other, Exterior2nd, MasVnrType, Foundation, SaleType and SaleCondition
- Converted fields of some Features to 0 or 1 based on the understanding of the dataset and a little bit research. Those features with the changing reasons are given below
  - IsRegularLotShape : Field IR2 and IR3 don't appear that often, so just make a distinction between regular and irregular.
  - IsLandLevel : Most land slopes are gentle; treat the others as "not gentle".
  - IsElectricalSBrkr : Most properties use standard circuit breakers.
  - IsGarageDetached : About 2/3rd have an attached garage.
  - IsPavedDrive : Most have a paved drive. Treat dirt/gravel and partial pavement as "not paved".
  - HasShed : The only interesting "misc. feature" is the presence of a shed.
  - Remodeled : If YearRemodAdd != YearBuilt, then a remodeling took place at some point.
  - RecentRemodel : Did a remodeling happen in the year the house was sold?
  - VeryNewHouse : Was this house sold in the year it was built?
  - sofe other features dont need to describe they are self explanatory Has2ndFloor , HasMasVnr , HasWoodDeck .HasOpenPorch ,HasEnclosedPorch ,Has3SsnPorch , HasScreenPorch
- Simplifications of existing features into bad/average/good. Features : SimplOverallQual, SimplOverallCond, SimplPoolQC ,SimplGarageCond, SimplGarageQual, SimplFireplaceQu ,SimplFunctional ,SimplKitchenQual, SimplHeatingQC ,SimplBsmtFinType1, SimplBsmtFinType2 ,SimplBsmtCond , SimplBsmtQual ,SimplExterCond ,SimplExterQual.
- mapped neighborhood based on their quality.The mapping is as followed:

```
        "MeadowV" : 0,  #  88000
        "IDOTRR" : 1,   # 103000
        "BrDale" : 1,   # 106000
        "OldTown" : 1,  # 119000
        "Edwards" : 1,  # 119500
        "BrkSide" : 1,  # 124300
        "Sawyer" : 1,   # 135000
        "Blueste" : 1,  # 137500
```

In [16]:
```
train_processed = data_process(train)
test_processed = data_process(test)

print("shape of train :" , train_processed.shape)
print("shape of test :" , test_processed.shape)
```

```
shape of train : (1458, 111)
shape of test : (1459, 111)
```

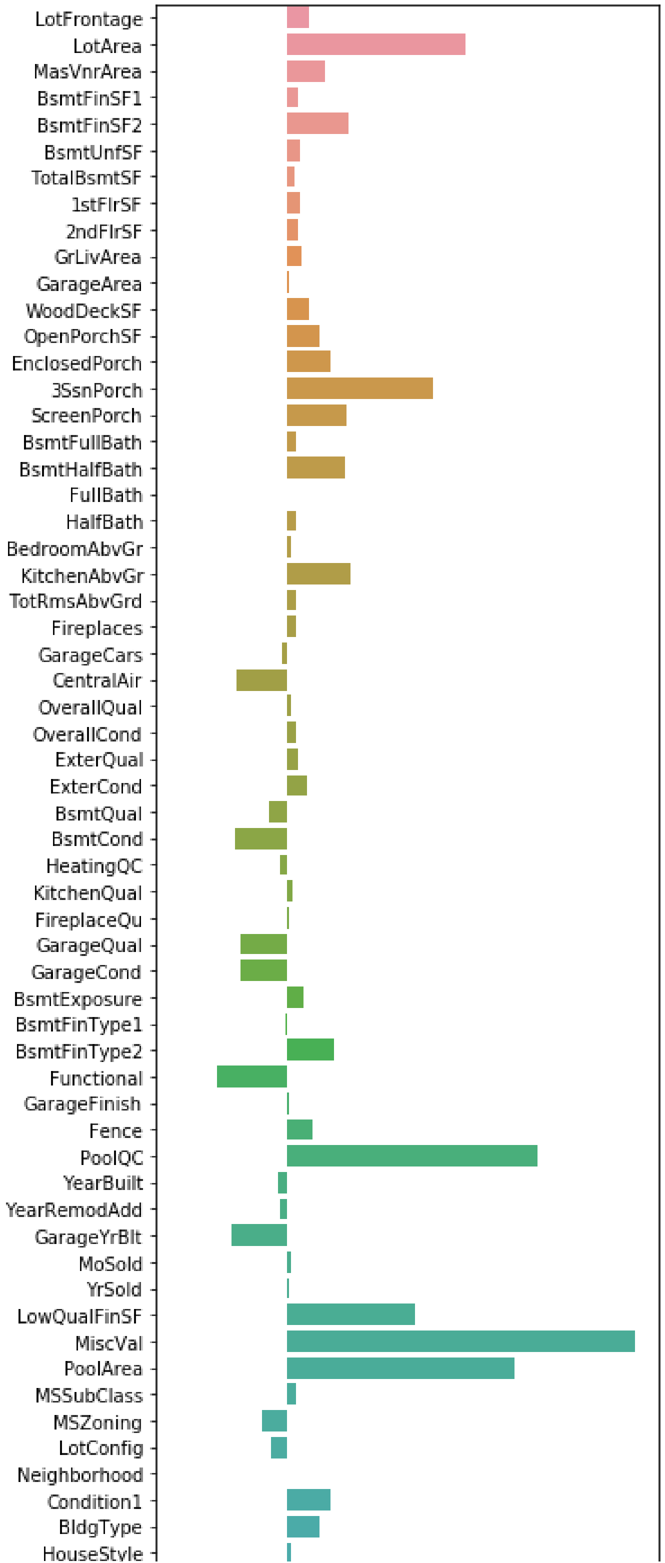
Keeping NeighborhoodBin into a temporary DataFrame because we want to use the unscaled version later on (to one-hot encode it).

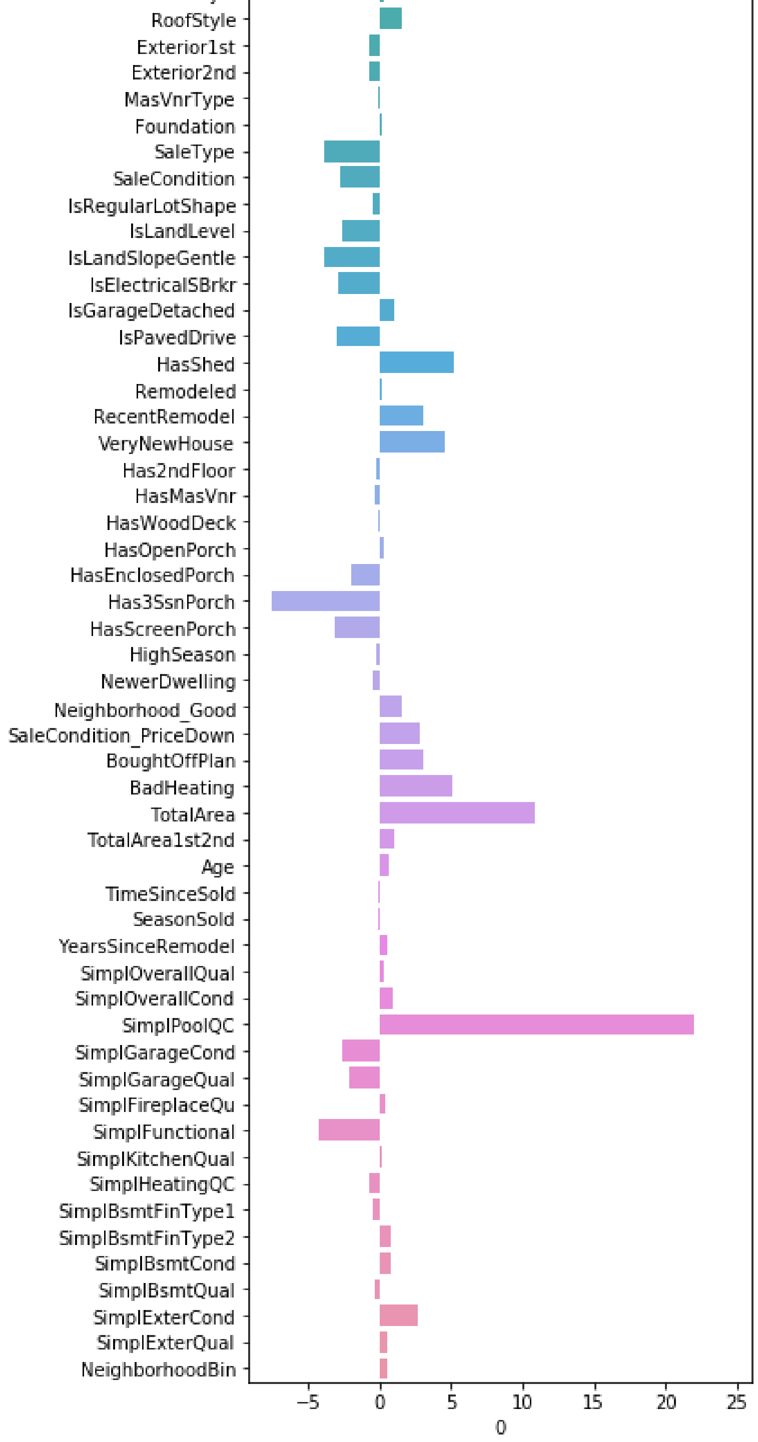
In [17]:
```
# Keeping NeighborhoodBin into a temporary DataFrame because we want to use the
# unscaled version later on (to one-hot encode it).
neighborhood_bin_train = pd.DataFrame(index = train.index)
neighborhood_bin_train["NeighborhoodBin"] = train_processed["NeighborhoodBin"]
neighborhood_bin_test = pd.DataFrame(index = test.index)
neighborhood_bin_test["NeighborhoodBin"] = test_processed["NeighborhoodBin"]
```

Skewness & Normalization

skewness train set

In [18]:
```
from scipy.stats import skew
import seaborn as sns
numeric_features = train_processed.dtypes[train_processed.dtypes != "object"].index

skewness = train_processed[numeric_features].skew(axis=0 , skipna =True)
skewness = pd.DataFrame(skewness)
plt.figure(figsize=[5,30])
# skw = sns.load_dataset(skewness)
ax = sns.barplot( y= skewness.index , x=skewness[0] , data = skewness)
plt.show()
# print('skew: ',train_processed[numeric_features].skew())
```

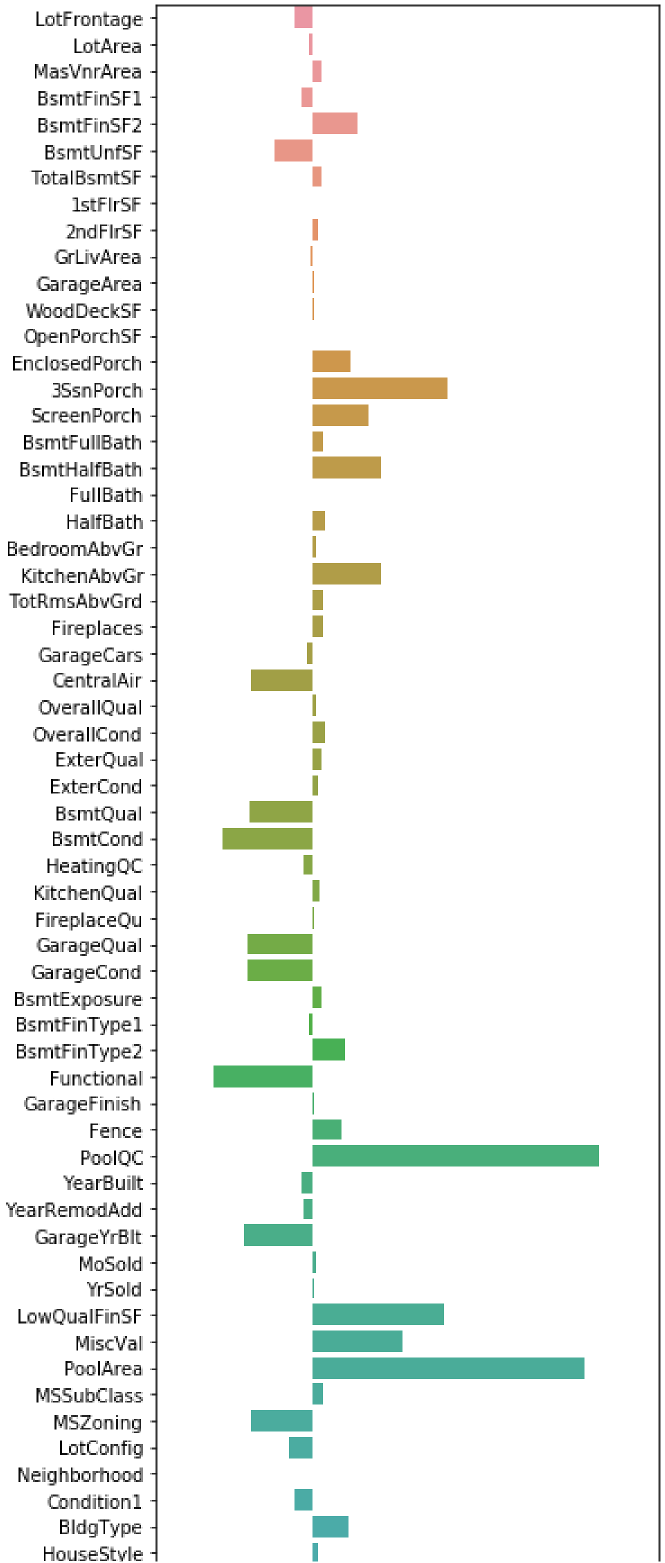

LotFrontage
LotArea
MasVnrArea
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
GrLivArea
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
ScreenPorch
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageCars
CentralAir
OverallQual
OverallCond
ExterQual
ExterCond
BsmtQual
BsmtCond
HeatingQC
KitchenQual
FireplaceQu
GarageQual
GarageCond
BsmtExposure
BsmtFinType1
BsmtFinType2
Functional
GarageFinish
Fence
PoolQC
YearBuilt
YearRemodAdd
GarageYrBlt
MoSold
YrSold
LowQualFinSF
MiscVal
PoolArea
MSSubClass
MSZoning
LotConfig
Neighborhood
Condition1
BldgType
HouseStyle


RoofStyle
Exterior1st
Exterior2nd
MasVnrType
Foundation
SaleType
SaleCondition
IsRegularLotShape
IsLandLevel
IsLandSlopeGentle
IsElectricalSBrkr
IsGarageDetached
IsPavedDrive
HasShed
Remodeled
RecentRemodel
VeryNewHouse
Has2ndFloor
HasMasVnr
HasWoodDeck
HasOpenPorch
HasEnclosedPorch
Has3SsnPorch
HasScreenPorch
HighSeason
NewerDwelling
Neighborhood_Good
SaleCondition_PriceDown
BoughtOffPlan
BadHeating
TotalArea
TotalArea1st2nd
Age
TimeSinceSold
SeasonSold
YearsSinceRemodel
SimplOverallQual
SimplOverallCond
SimplPoolQC
SimplGarageCond
SimplGarageQual
SimplFireplaceQu
SimplFunctional
SimplKitchenQual
SimplHeatingQC
SimplBsmtFinType1
SimplBsmtFinType2
SimplBsmtCond
SimplBsmtQual
SimplExterCond
SimplExterQual
NeighborhoodBin
−5
0
5
10
15
20
25
0

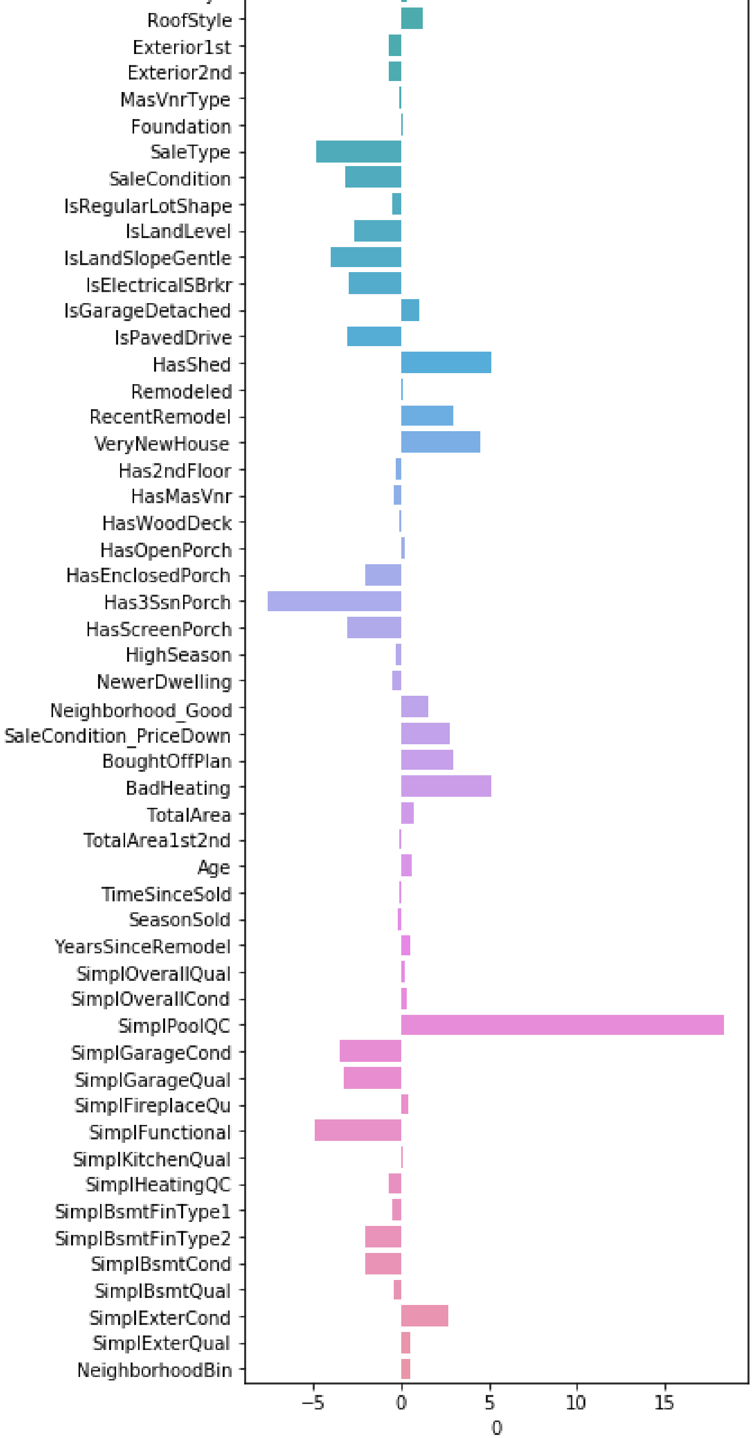
In [19]:
```python
numeric_features = train_processed.dtypes[train_processed.dtypes != "object"].index

# Transform the skewed numeric features by taking log(feature + 1).
# This will make the features more normal.
from scipy.stats import skew

skewed = train_processed[numeric_features].apply(lambda x: skew(x.dropna().astype(float)))
skewed = skewed[(skewed < -0.75) | (skewed > 0.75)]
skewed = skewed.index

train_processed[skewed] = np.log1p(train_processed[skewed])


# Additional processing: scale the data.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled = scaler.fit_transform(train_processed[numeric_features])

for i, col in enumerate(numeric_features):
    train_processed[col] = scaled[:, i]
```

```
/home/navid/anaconda3/envs/tf/lib/python3.6/site-packages/sklearn/preprocessing/data.py:645: DataConversionWarning: Data with inp
ut dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/navid/anaconda3/envs/tf/lib/python3.6/site-packages/sklearn/base.py:464: DataConversionWarning: Data with input dtype int6
4, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [20]:
```
from scipy.stats import skew
numeric_features = train_processed.dtypes[train_processed.dtypes != "object"].index
skewness = train_processed[numeric_features].skew(axis=0 , skipna =True)
skewness = pd.DataFrame(skewness)
plt.figure(figsize=[5,30])
# skw = sns.load_dataset(skewness)
ax = sns.barplot( y= skewness.index , x=skewness[0] , data = skewness)
plt.show()
```

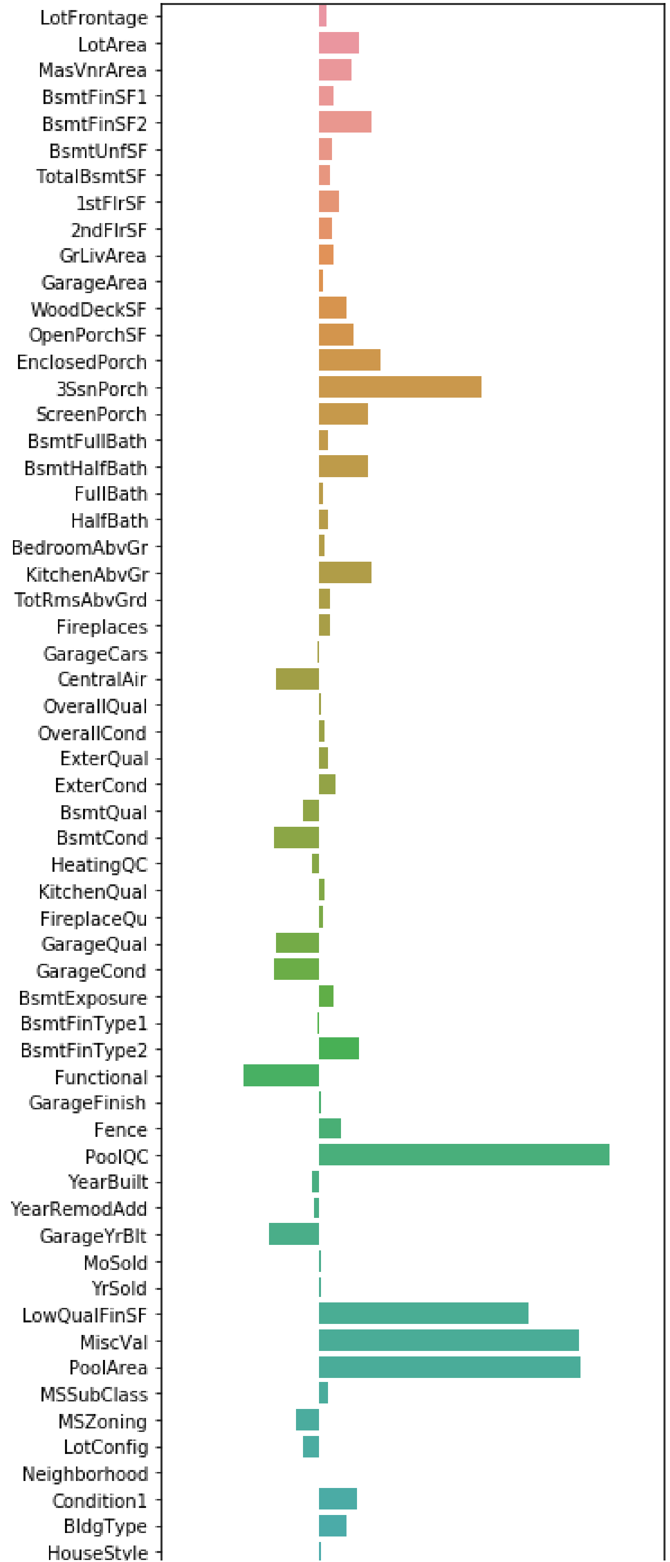


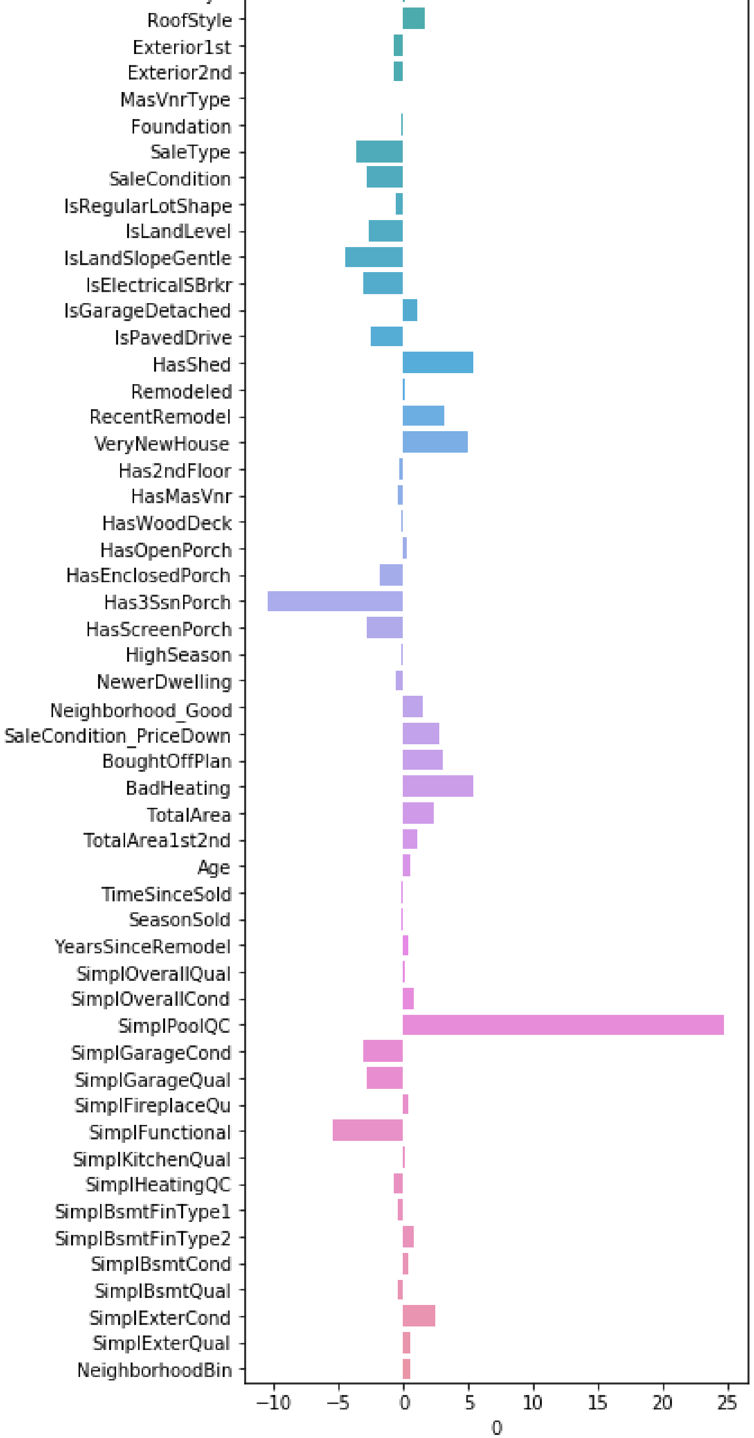
LotFrontage
LotArea
MasVnrArea
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
GrLivArea
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
ScreenPorch
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageCars
CentralAir
OverallQual
OverallCond
ExterQual
ExterCond
BsmtQual
BsmtCond
HeatingQC
KitchenQual
FireplaceQu
GarageQual
GarageCond
BsmtExposure
BsmtFinType1
BsmtFinType2
Functional
GarageFinish
Fence
PoolQC
YearBuilt
YearRemodAdd
GarageYrBlt
MoSold
YrSold
LowQualFinSF
MiscVal
PoolArea
MSSubClass
MSZoning
LotConfig
Neighborhood
Condition1
BldgType
HouseStyle


RoofStyle
Exterior1st
Exterior2nd
MasVnrType
Foundation
SaleType
SaleCondition
IsRegularLotShape
IsLandLevel
IsLandSlopeGentle
IsElectricalSBrkr
IsGarageDetached
IsPavedDrive
HasShed
Remodeled
RecentRemodel
VeryNewHouse
Has2ndFloor
HasMasVnr
HasWoodDeck
HasOpenPorch
HasEnclosedPorch
Has3SsnPorch
HasScreenPorch
HighSeason
NewerDwelling
Neighborhood_Good
SaleCondition_PriceDown
BoughtOffPlan
BadHeating
TotalArea
TotalArea1st2nd
Age
TimeSinceSold
SeasonSold
YearsSinceRemodel
SimplOverallQual
SimplOverallCond
SimplPoolQC
SimplGarageCond
SimplGarageQual
SimplFireplaceQu
SimplFunctional
SimplKitchenQual
SimplHeatingQC
SimplBsmtFinType1
SimplBsmtFinType2
SimplBsmtCond
SimplBsmtQual
SimplExterCond
SimplExterQual
NeighborhoodBin
−5
0
5
10
15
0

Test Skewness

In [21]:
```
numeric_features = test_processed.dtypes[train_processed.dtypes != "object"].index
skewness = test_processed[numeric_features].skew(axis=0 , skipna =True)
skewness = pd.DataFrame(skewness)

plt.figure(figsize=[5,30])
# skw = sns.load_dataset(skewness)
ax = sns.barplot( y= skewness.index , x=skewness[0] , data = skewness)
plt.show()
# print('skew: ',test_processed[numeric_features].skew())
```


LotFrontage
LotArea
MasVnrArea
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
GrLivArea
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
ScreenPorch
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageCars
CentralAir
OverallQual
OverallCond
ExterQual
ExterCond
BsmtQual
BsmtCond
HeatingQC
KitchenQual
FireplaceQu
GarageQual
GarageCond
BsmtExposure
BsmtFinType1
BsmtFinType2
Functional
GarageFinish
Fence
PoolQC
YearBuilt
YearRemodAdd
GarageYrBlt
MoSold
YrSold
LowQualFinSF
MiscVal
PoolArea
MSSubClass
MSZoning
LotConfig
Neighborhood
Condition1
BldgType
HouseStyle

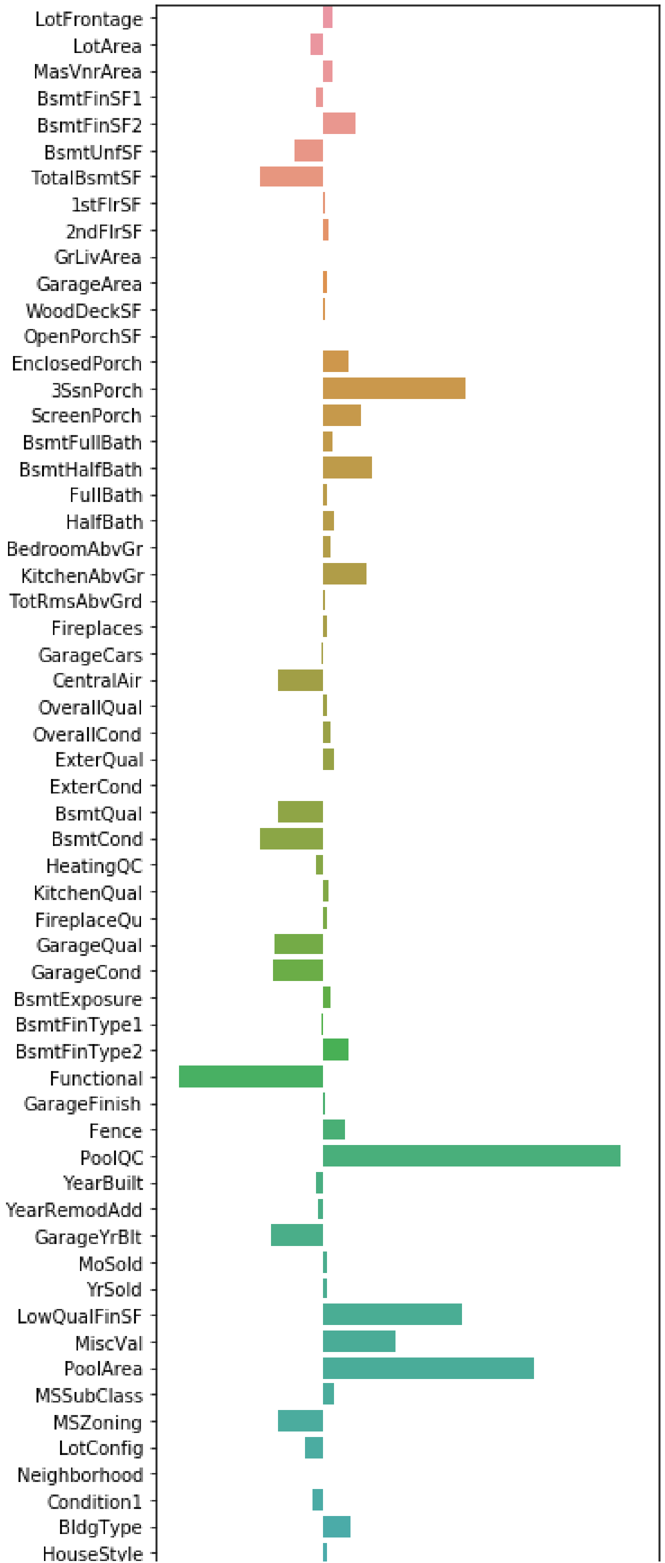

RoofStyle
Exterior1st
Exterior2nd
MasVnrType
Foundation
SaleType
SaleCondition
IsRegularLotShape
IsLandLevel
IsLandSlopeGentle
IsElectricalSBrkr
IsGarageDetached
IsPavedDrive
HasShed
Remodeled
RecentRemodel
VeryNewHouse
Has2ndFloor
HasMasVnr
HasWoodDeck
HasOpenPorch
HasEnclosedPorch
Has3SsnPorch
HasScreenPorch
HighSeason
NewerDwelling
Neighborhood_Good
SaleCondition_PriceDown
BoughtOffPlan
BadHeating
TotalArea
TotalArea1st2nd
Age
TimeSinceSold
SeasonSold
YearsSinceRemodel
SimplOverallQual
SimplOverallCond
SimplPoolQC
SimplGarageCond
SimplGarageQual
SimplFireplaceQu
SimplFunctional
SimplKitchenQual
SimplHeatingQC
SimplBsmtFinType1
SimplBsmtFinType2
SimplBsmtCond
SimplBsmtQual
SimplExterCond
SimplExterQual
NeighborhoodBin
−10
−5
0
5
10
15
20
25
0

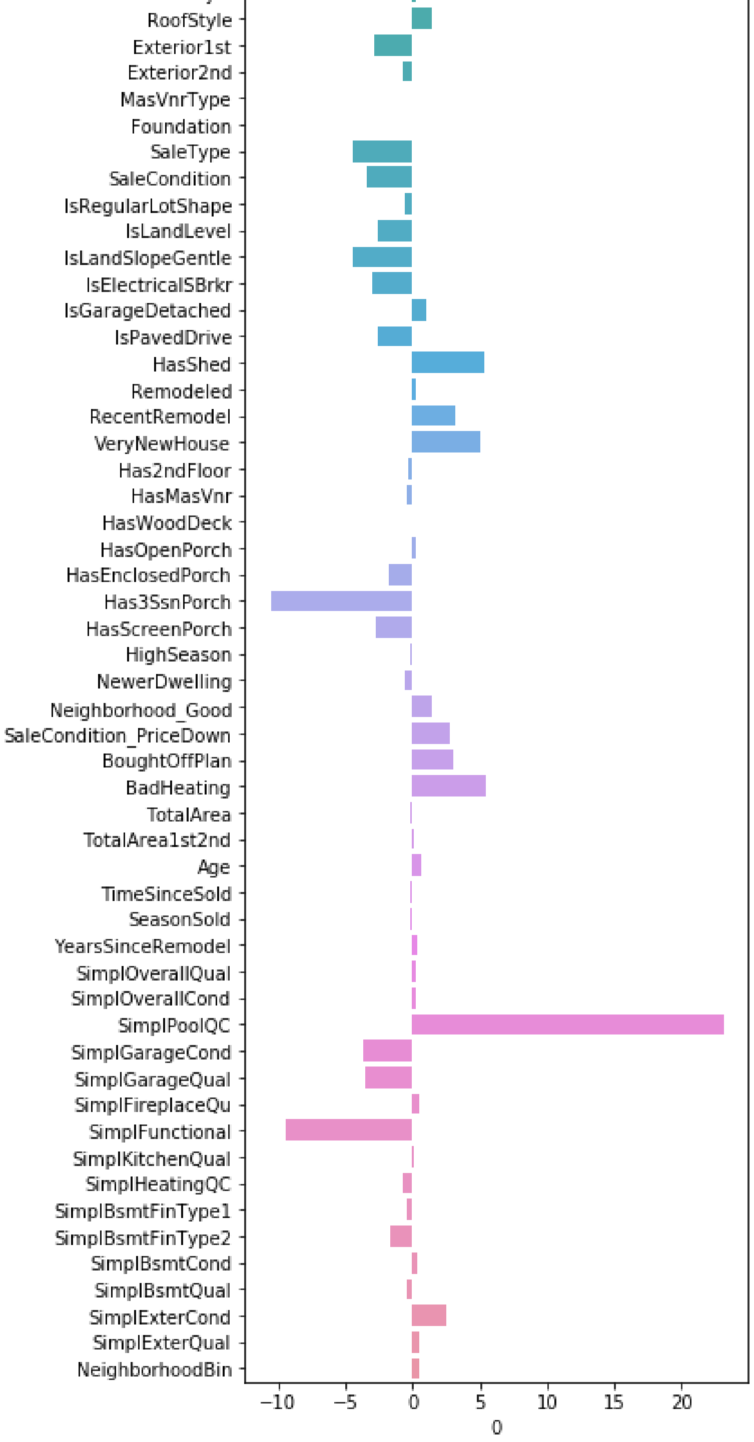
In [22]:

```
numeric_features = test_processed.dtypes[train_processed.dtypes != "object"].index

# Transform the skewed numeric features by taking log(feature + 1).
# This will make the features more normal.
from scipy.stats import skew

skewed = test_processed[numeric_features].apply(lambda x: skew(x.dropna().astype(float)))
skewed = skewed[(skewed < -0.75) | (skewed > 0.75)]
skewed = skewed.index

test_processed[skewed] = np.log1p(test_processed[skewed])

# Additional processing: scale the data.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaled = scaler.fit_transform(test_processed[numeric_features])
for i, col in enumerate(numeric_features):
    test_processed[col] = scaled[:, i]
```

```
/home/navid/anaconda3/envs/tf/lib/python3.6/site-packages/sklearn/preprocessing/data.py:645: DataConversionWarning: Data with inp
ut dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/navid/anaconda3/envs/tf/lib/python3.6/site-packages/sklearn/base.py:464: DataConversionWarning: Data with input dtype int6
4, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [23]:
```
numeric_features = test_processed.dtypes[train_processed.dtypes != "object"].index
skewness = test_processed[numeric_features].skew(axis=0 , skipna =True)
skewness = pd.DataFrame(skewness)

plt.figure(figsize=[5,30])
# skw = sns.load_dataset(skewness)
ax = sns.barplot( y= skewness.index , x=skewness[0] , data = skewness)
plt.show()
# print('skew: ',test_processed[numeric_features].skew())
```


LotFrontage
LotArea
MasVnrArea
BsmtFinSF1
BsmtFinSF2
BsmtUnfSF
TotalBsmtSF
1stFlrSF
2ndFlrSF
GrLivArea
GarageArea
WoodDeckSF
OpenPorchSF
EnclosedPorch
3SsnPorch
ScreenPorch
BsmtFullBath
BsmtHalfBath
FullBath
HalfBath
BedroomAbvGr
KitchenAbvGr
TotRmsAbvGrd
Fireplaces
GarageCars
CentralAir
OverallQual
OverallCond
ExterQual
ExterCond
BsmtQual
BsmtCond
HeatingQC
KitchenQual
FireplaceQu
GarageQual
GarageCond
BsmtExposure
BsmtFinType1
BsmtFinType2
Functional
GarageFinish
Fence
PoolQC
YearBuilt
YearRemodAdd
GarageYrBlt
MoSold
YrSold
LowQualFinSF
MiscVal
PoolArea
MSSubClass
MSZoning
LotConfig
Neighborhood
Condition1
BldgType
HouseStyle


RoofStyle
Exterior1st
Exterior2nd
MasVnrType
Foundation
SaleType
SaleCondition
IsRegularLotShape
IsLandLevel
IsLandSlopeGentle
IsElectricalSBrkr
IsGarageDetached
IsPavedDrive
HasShed
Remodeled
RecentRemodel
VeryNewHouse
Has2ndFloor
HasMasVnr
HasWoodDeck
HasOpenPorch
HasEnclosedPorch
Has3SsnPorch
HasScreenPorch
HighSeason
NewerDwelling
Neighborhood_Good
SaleCondition_PriceDown
BoughtOffPlan
BadHeating
TotalArea
TotalArea1st2nd
Age
TimeSinceSold
SeasonSold
YearsSinceRemodel
SimplOverallQual
SimplOverallCond
SimplPoolQC
SimplGarageCond
SimplGarageQual
SimplFireplaceQu
SimplFunctional
SimplKitchenQual
SimplHeatingQC
SimplBsmtFinType1
SimplBsmtFinType2
SimplBsmtCond
SimplBsmtQual
SimplExterCond
SimplExterQual
NeighborhoodBin
−10
−5
0
5
10
15
20
0

Additional processing to scale the data.

One hot encoding

To encode categorical integer features as a one-hot numeric array we are using one hot encoding. This will transform each value of catagories into a features and make those a column value of dataframe. Finally put binary values in the rows of those column.

In [24]:
```
# for example:

#  ╔══════════════════╦═══════════════════╦═══════╗
#  ║ CompanyName Categoricalvalue ║ Price  ║
#  ╠══════════════════╬═══════════════════╣═══════║
#  ║ VW          ╬      1          ║ 20000  ║
#  ║ Acura       ╬      2          ║ 10011  ║
#  ║ Honda       ╬      3          ║ 50000  ║
#  ║ Honda       ╬      3          ║ 10000  ║
#  ╚══════════════════╩═══════════════════╩═══════╝

# converting it to one Hot encoding:

#  ╔═══════╦═══════╦═══════╦═══════╗
#  ║ VW  ║ Acura║ Honda║ Price  ║
#  ╠═══════╬═══════╬═══════╬═══════╣
#  ║ 1   ╬ 0    ╬ 0    ║ 20000  ║
#  ║ 0   ╬ 1    ╬ 0    ║ 10011  ║
#  ║ 0   ╬ 0    ╬ 1    ║ 50000  ║
#  ║ 0   ╬ 0    ╬ 1    ║ 10000  ║
#  ╚═══════╩═══════╩═══════╩═══════╝

# refrence: https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f
```

In this section at first we merge train and test data (variable name predictor_cols and predictor_cols_test). We did it because there is some features in train data which is missing in test data again same thing can happen for test data too.

In [25]:
```
# Convert categorical features using one-hot encoding.
def onehot(onehot_df, df, column_name, fill_na, drop_name):
    onehot_df[column_name] = df[column_name]
    if fill_na is not None:
        onehot_df[column_name].fillna(fill_na, inplace=True)

    dummies = pd.get_dummies(onehot_df[column_name], prefix="_" + column_name)

    onehot_df = onehot_df.join(dummies)
    onehot_df = onehot_df.drop([column_name], axis=1)
    return onehot_df
```

performing one hot

In [26]:
```
def proceed_onehot(df):
    onehot_df = pd.DataFrame(index = df.index)

    onehot_df = onehot(onehot_df, df, "MSSubClass", None, "40")
    onehot_df = onehot(onehot_df, df, "MSZoning", "RL", "RH")
    onehot_df = onehot(onehot_df, df, "LotConfig", None, "FR3")
    onehot_df = onehot(onehot_df, df, "Neighborhood", None, "OldTown")
    onehot_df = onehot(onehot_df, df, "Condition1", None, "RRNe")
    onehot_df = onehot(onehot_df, df, "BldgType", None, "2fmCon")
    onehot_df = onehot(onehot_df, df, "HouseStyle", None, "1.5Unf")
    onehot_df = onehot(onehot_df, df, "RoofStyle", None, "Shed")
    onehot_df = onehot(onehot_df, df, "Exterior1st", "VinylSd", "CBlock")
    onehot_df = onehot(onehot_df, df, "Exterior2nd", "VinylSd", "CBlock")
    onehot_df = onehot(onehot_df, df, "Foundation", None, "Wood")
    onehot_df = onehot(onehot_df, df, "SaleType", "WD", "Oth")
    onehot_df = onehot(onehot_df, df, "SaleCondition", "Normal", "AdjLand")

    # Fill in missing MasVnrType for rows that do have a MasVnrArea.
    temp_df = df[["MasVnrType", "MasVnrArea"]].copy()
    idx = (df["MasVnrArea"] != 0) & ((df["MasVnrType"] == "None") | (df["MasVnrType"].isnull()))
    temp_df.loc[idx, "MasVnrType"] = "BrkFace"
    onehot_df = onehot(onehot_df, temp_df, "MasVnrType", "None", "BrkCmn")

    # Also add the booleans from calc_df as dummy variables.
    onehot_df = onehot(onehot_df, df, "LotShape", None, "IR3")
    onehot_df = onehot(onehot_df, df, "LandContour", None, "Low")
    onehot_df = onehot(onehot_df, df, "LandSlope", None, "Sev")
    onehot_df = onehot(onehot_df, df, "Electrical", "SBrkr", "FuseP")
    onehot_df = onehot(onehot_df, df, "GarageType", "None", "CarPort")
    onehot_df = onehot(onehot_df, df, "PavedDrive", None, "P")
    onehot_df = onehot(onehot_df, df, "MiscFeature", "None", "Othr")

    # Features we can probably ignore (but want to include anyway to see
    # if they make any positive difference).
    # Definitely ignoring Utilities: all records are "AllPub", except for
    # one "NoSeWa" in the train set and 2 NA in the test set.
    onehot_df = onehot(onehot_df, df, "Street", None, "Grvl")
    onehot_df = onehot(onehot_df, df, "Alley", "None", "Grvl")
    onehot_df = onehot(onehot_df, df, "Condition2", None, "PosA")
    onehot_df = onehot(onehot_df, df, "RoofMatl", None, "WdShake")
    onehot_df = onehot(onehot_df, df, "Heating", None, "Wall")

    # I have these as numerical variables too.
    onehot_df = onehot(onehot_df, df, "ExterQual", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "ExterCond", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "BsmtQual", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "BsmtCond", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "HeatingQC", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "KitchenQual", "TA", "Ex")
    onehot_df = onehot(onehot_df, df, "FireplaceQu", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "GarageQual", "None", "Ex")
```

```
    onehot_df = onehot(onehot_df, df, "GarageCond", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "PoolQC", "None", "Ex")
    onehot_df = onehot(onehot_df, df, "BsmtExposure", "None", "Gd")
    onehot_df = onehot(onehot_df, df, "BsmtFinType1", "None", "GLQ")
    onehot_df = onehot(onehot_df, df, "BsmtFinType2", "None", "GLQ")
    onehot_df = onehot(onehot_df, df, "Functional", "Typ", "Typ")
    onehot_df = onehot(onehot_df, df, "GarageFinish", "None", "Fin")
    onehot_df = onehot(onehot_df, df, "Fence", "None", "MnPrv")
    onehot_df = onehot(onehot_df, df, "MoSold", None, None)

    # Divide up the years between 1871 and 2010 in slices of 20 years.
    year_map = pd.concat(pd.Series("YearBin" + str(i+1), index=range(1871+i*20,1891+i*20)) for i in range(0, 7))

    yearbin_df = pd.DataFrame(index = df.index)
    yearbin_df["GarageYrBltBin"] = df.GarageYrBlt.map(year_map)
    yearbin_df["GarageYrBltBin"].fillna("NoGarage", inplace=True)

    yearbin_df["YearBuiltBin"] = df.YearBuilt.map(year_map)
    yearbin_df["YearRemodAddBin"] = df.YearRemodAdd.map(year_map)

    onehot_df = onehot(onehot_df, yearbin_df, "GarageYrBltBin", None, None)
    onehot_df = onehot(onehot_df, yearbin_df, "YearBuiltBin", None, None)
    onehot_df = onehot(onehot_df, yearbin_df, "YearRemodAddBin", None, None)

    return onehot_df

# Add the one-hot encoded categorical features.
onehot_df = proceed_onehot(train)
onehot_df = onehot(onehot_df, neighborhood_bin_train, "NeighborhoodBin", None, None)
train_processed = train_processed.join(onehot_df)
```

These onehot columns are missing in the test data, so drop them from the training data or we might overfit on them.

In [27]:
```
drop_cols = [
                "_Exterior1st_ImStucc", "_Exterior1st_Stone",
                "_Exterior2nd_Other","_HouseStyle_2.5Fin",

                "_RoofMatl_Membran", "_RoofMatl_Metal", "_RoofMatl_Roll",
                "_Condition2_RRAe", "_Condition2_RRAn", "_Condition2_RRNn",
                "_Heating_Floor", "_Heating_OthW",

                "_Electrical_Mix",
                "_MiscFeature_TenC",
                "_GarageQual_Ex", "_PoolQC_Fa"
            ]
train_processed.drop(drop_cols, axis=1, inplace=True)
```

In [28]:
```
onehot_df = proceed_onehot(test)
onehot_df = onehot(onehot_df, neighborhood_bin_test, "NeighborhoodBin", None, None)
test_processed = test_processed.join(onehot_df)
```

This column is missing in the training data. There is only one example with this value in the test set. So just drop it.

In [29]:
```
test_processed.drop(["_MSSubClass_150"], axis=1, inplace=True)
```

Drop these columns. They are either not very helpful or they cause overfitting.

In [30]:
```
drop_cols = [
    "_Condition2_PosN",    # only two are not zero
    "_MSZoning_C (all)",
    "_MSSubClass_160",
]
train_processed.drop(drop_cols, axis=1, inplace=True)
test_processed.drop(drop_cols, axis=1, inplace=True)
```

log transform

We take the log here because the error metric is between the log of the SalePrice and the log of the predicted price. That does mean we need to exp() the prediction to get an actual sale price.

In [31]:
```
target = pd.DataFrame(index = train_processed.index, columns=["SalePrice"])
target["SalePrice"] = np.log(train["SalePrice"])
# train_processed.drop(["SalePrice"], axis=1, inplace=True)

print("Training set size:", train_processed.shape)
print("Test set size:", test_processed.shape)
```

```
Training set size: (1458, 403)
Test set size: (1459, 403)
```

Split Data for training and testing

In [32]:
```
X_train, X_test, y_train, y_test = train_test_split(train_processed,
                                                     target,
#                                                      train_size = 0.99,
                                                     test_size = 0.2,
                                                     random_state = 0)
```

In [33]:
```python
prediction_dict = dict()
submit_prediction_dict = dict()

submit = False
save_score = False

if submit :
    X_train = train_processed
    y_train = target
else:
    X_train = X_train
    y_train = y_train
```

Testing different models

Random Forest Regressor

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

In [34]:
```python
my_model = RandomForestRegressor(n_estimators=500,n_jobs=-1)


my_model.fit(X_train, y_train)
prediction = my_model.predict(X_test)
if submit:
    submit_prediction = my_model.predict(test_processed)
    submit_prediction_dict['Random Forest Regressor'] = submit_prediction

prediction_dict['Random Forest Regressor'] = prediction

print('root mean absolute error: ',rmse(y_test, prediction))
print('accuracy score: ',  r2_score(np.array(y_test),prediction) )
```

```
/home/navid/anaconda3/envs/tf/lib/python3.6/site-packages/ipykernel_launcher.py:4: DataConversionWarning: A column-vector y was p
assed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  after removing the cwd from sys.path.

ann root mean absolute error:  0.12391751965552095
accuracy score:  0.9093931764105156
```

DecisionTree

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

In [35]:
```
from sklearn.tree import DecisionTreeRegressor
my_model = DecisionTreeRegressor()

my_model.fit(X_train, y_train)
prediction = my_model.predict(X_test)
prediction_dict['DecisionTree'] = prediction
if submit:
    submit_prediction = my_model.predict(test_processed)
    submit_prediction_dict['DecisionTree'] = submit_prediction

print('root mean absolute error: ',rmse(y_test, prediction))
print('accuracy score: ',  r2_score(np.array(y_test),prediction) )
```

```
ann root mean absolute error:  0.19407309037420048
accuracy score:  0.7777580061513429
```

Xgboost

XGBoost stands for eXtreme Gradient Boosting. It is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

In [36]:
```
from xgboost import XGBRegressor
my_model = XGBRegressor(n_estimators=500, learning_rate=0.05)

my_model.fit(X_train, y_train)
prediction = my_model.predict(X_test)
prediction_dict['Xgboost'] = prediction

if submit:
    submit_prediction = my_model.predict(test_processed)
    submit_prediction_dict['Xgboost'] = submit_prediction


print('root mean absolute error: ',rmse(y_test, prediction))
print('accuracy score: ',  r2_score(np.array(y_test),prediction) )
```

```
ann root mean absolute error:  0.10926796451065869
accuracy score:  0.9295499691663187
```

Lasso

Lasso (least absolute shrinkage and selection operator; also Lasso or LASSO) is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces.Lasso was originally formulated for least squares models and this simple case reveals a substantial amount about the behavior of the estimator, including its relationship to ridge regression and best subset selection and the connections between lasso coefficient estimates and so-called soft thresholding. It also reveals that (like standard linear regression) the coefficient estimates need not be unique if covariates are collinear.

In [50]:
```
from sklearn.linear_model import Lasso
my_model = Lasso(alpha=5e-3, max_iter=50000)


my_model.fit(X_train, y_train)
prediction = my_model.predict(X_test)
prediction_dict['Lasso'] = prediction

if submit:
    submit_prediction = my_model.predict(test_processed)
    submit_prediction_dict['Lasso'] = submit_prediction

print(' root mean absolute error: ',rmse(y_test, prediction))
print('accuracy score: ',  r2_score(np.array(y_test),prediction) )
```

```
 root mean absolute error:  0.10794617678311977
accuracy score:  0.9312440935471524
```

In the above model alpha is Constant that multiplies the L1 term. For numerical reason we cant set alpha to 0 but keeping alpha low provides good accuracy for out dataset. I have found 5e-4 provides good accuracy.

for 5e-5: root mean absolute error: 0.10973737757187135 accuracy score: 0.9289433650407954

for 1e-5: root mean absolute error: 0.11426822609093419 accuracy score: 0.9229546464396043

for 1e-3: root mean absolute error: 0.10466883446067998 accuracy score: 0.9353556969018821

for 1e-4: root mean absolute error: 0.10658498063306822 accuracy score: 0.9329671780226085

for 5e-3: root mean absolute error: 0.10794617678311977 accuracy score: 0.9312440935471524

ANN

Theory and Basics:

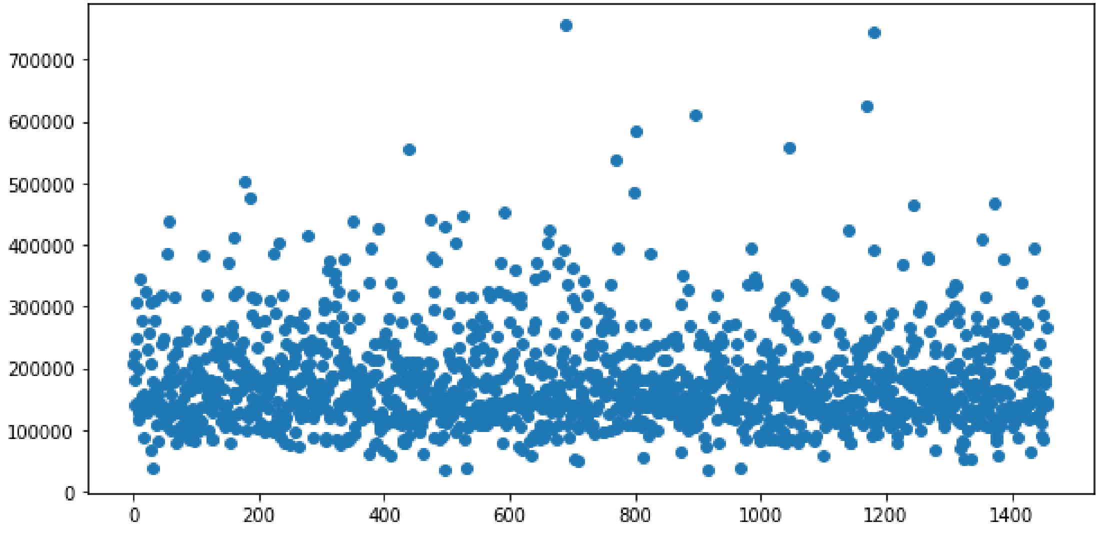
An Artificial Neurol Network (ANN) is a computational model. It is based on the structure and functions of biological neural networks. It works like the way human brain processes information. ANN includes a large number of connected processing units that work together to process information. They also generate meaningful results from it.

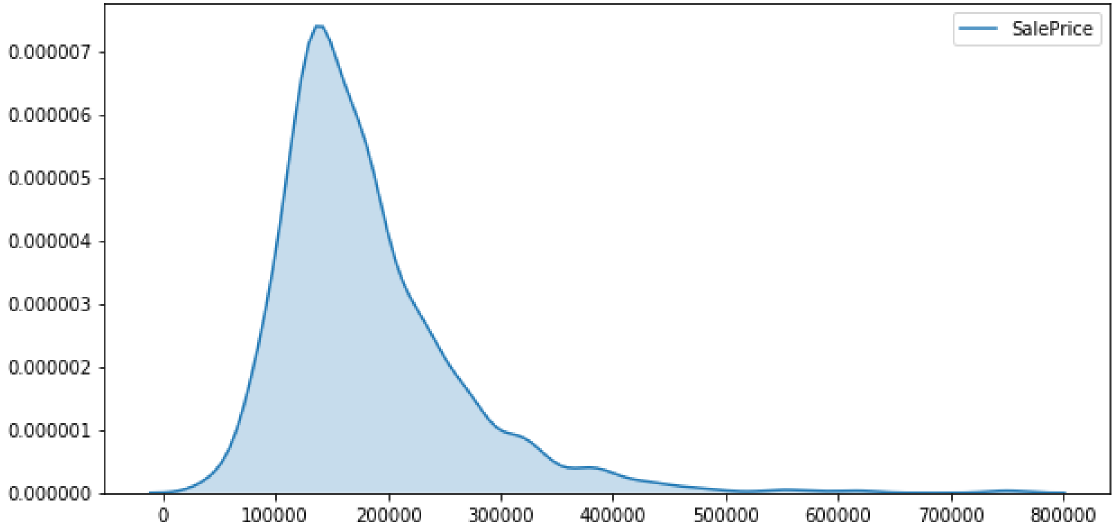
An artificial neuron is a mathematical function conceived as a model of biological neurons, a neural network. Usually each input is separately weighted, and the sum is passed through a non-linear function known as an activation function or transfer function.

The artificial Neural network is typically organized in layers. Layers are being made up of many interconnected ‘nodes’ which contain an ‘activation function’. A neural network may contain the following 3 layers:

- **Input layer** The purpose of the input layer is to receive as input the values of the explanatory attributes for each observation. Usually, the number of input nodes in an input layer is equal to the number of explanatory variables. ‘input layer’ presents the patterns to the network, which communicates to one or more ‘hidden layers’. The nodes of the input layer are passive, meaning they do not change the data. They receive a single value on their input and duplicate the value to their many outputs. From the input layer, it duplicates each value and sent to all the hidden nodes.
- **Hidden Layer** The Hidden layers apply given transformations to the input values inside the network. In this, incoming arcs that go from other hidden nodes or from input nodes connected to each node. It connects with outgoing arcs to output nodes or to other hidden nodes. In hidden layer, the actual processing is done via a system of weighted ‘connections’. There may be one or more hidden layers. The values entering a hidden node multiplied by weights, a set of predetermined numbers stored in the program. The weighted inputs are then added to produce a single number.
- **Output layer** The hidden layers then link to an ‘output layer‘. Output layer receives connections from hidden layers or from input layer. It returns an output value that corresponds to the prediction of the response variable. In classification problems, there is usually only one output node. The active nodes of the output layer combine and change the data to produce the output values.

In [35]:
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=[10,5])
plt.scatter(range(len(train)),list(train.SalePrice.values))
plt.show()
plt.figure(figsize=[10,5])
sns.kdeplot(train.SalePrice, shade= True)
plt.show()
```





In the above graph we can see that the price range is in a normal distribution. If we provide tf.random.normal while initializing the weight it should be more helpful for training. And this initialization should provide better validation with low amount of epoches. In my kaggle score rmse 0.123 is found through random normal while uniform distribution provided rmse 0.127 score. Again Uniform distribution takes 3 times more epoches to reach rmse score 0.127. But for uniform distribution no improvement cant found after 16000 epoch and for normal distribution no improvement can't found after 6000 epoch.

Target

By observing the span of the data and the data distribution we can conclude that logistic regression should perform well for this kind of problem. So we can safely say that starting with single neuron in a single hidden layer should perform well and we should look for simpler solution. Again from theoretical perspective single neurone and single layer ANN is nothing but a logistic regression and after adding layers and neurons we can regularize them so that they behave more like a logistic regression model and then we can tune parameter such a way so that it can handle little bit more complexity than a logistic regression. Finally my target is to make sure that it performs well as a logistic regression model and then improve it with more neuron/layers and proper tuning of parameters.

In [36]:
```
# log_df = pd.DataFrame(columns=['learning_rate', 'num_steps', 'beta1','beta2','beta3', 'hidden_1' , 'hidden_2', 'hidden_3','inpu
t_dim' , 'test_rmse_score', 'test_r2_score'])
# log_df.to_csv("diffrent_training_results.csv", index=False)
```

Ann parameters

The following ANN is build with 3 hidden layers. Output dimention is 1 because its a regration problem.

In [37]:
```
tf.reset_default_graph()
learning_rate = 0.1
num_steps = 6000
#for regularize weight matrix
beta1 = 0.1
beta2 = 0.00
beta3 = 0.00
beta4 = None

hidden_1 = 16
hidden_2 = 8
hidden_3 = 4
hidden_4 = None

# minimum_validation_cost is to control model saving locally
minimum_validation_cost = 0.0190000

input_dim = X_train.shape[1]  # Number of features
output_dim = 1              # Because it is a regression problem
#tf graph input
X_tf = tf.placeholder("float" )
y_tf = tf.placeholder("float" )
```

Initialization of weight and bias with random values

In [38]:
```
weights = {
    'w1': tf.Variable(tf.random_normal([input_dim, hidden_1])),
    'w2': tf.Variable(tf.random_normal([hidden_1, hidden_2])),
    'w3': tf.Variable(tf.random_normal([hidden_2, hidden_3])),
    'out': tf.Variable(tf.random_normal([hidden_3, output_dim]))
}
biases = {
    'b1': tf.Variable(tf.random_normal([hidden_1])),
    'b2': tf.Variable(tf.random_normal([hidden_2])),
    'b3': tf.Variable(tf.random_normal([hidden_3])),
    'out': tf.Variable(tf.random_normal([output_dim]))
}
```

```
WARNING:tensorflow:From /home/navid/anaconda3/envs/tf/lib/python3.6/site-packages/tensorflow/python/framework/op_def_library.py:2
63: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

Model

In the dataset Sales price are non negative number so our model is expected to return positive values so as a activation function I have used relu as it gives positive values. Again relu is easy to optimize because they are similar to linear units. The only difference is that a rectified linear unit outputs zero across half its domain. Thus derivatives through a rectified linear unit remain large whenever the unit is activate. The gradients are not only large but also consistent.

In [39]:
```
def ann_model(X_val):
  # Hidden layers
    layer_1 = tf.add(tf.matmul(X_val, weights['w1']), biases['b1'])
    layer_1 = tf.nn.relu(layer_1)


    layer_2 = tf.add(tf.matmul(layer_1, weights['w2']), biases['b2'])
    layer_2 = tf.nn.relu(layer_2)

    layer_3 = tf.add(tf.matmul(layer_2, weights['w3']), biases['b3'])
    layer_3 = tf.nn.relu(layer_3)


    # Output layer
    layer_out = tf.add(tf.matmul(layer_3, weights['out']), biases['out'])

    return layer_out
```

For optimization I have used Adam optimizer. Adam derives from phrase “adaptive moments”. Its a varient of RMSProp. I have used adam instead of RMSProp for couple of reasons. First, in Adam, momentum is incorporated directly as an estimate of the first-order moment (with exponential weighting) of the gradient. The most straightforward way to add momentum to RMSProp is to apply momentum to the rescaled gradients. The use of momentum in combination with rescaling does not have a clear theoretical motivation. Second, Adam includes bias corrections to the estimates of both the first-order moments (the momentumterm) and the (uncentered) second-order moments to account for their initializationat the origin. RMSProp also incorporates an estimate of the (uncentered) second-order moment; however, it lacks the correction factor. Thus,unlike in Adam, the RMSProp second-order moment estimate may have high bias early in training. Adam is generally regarded as being fairly robust to the choice of hyperparameters, though the learning rate sometimes needs to be changed from the suggested default. Usually default rate is .001 but for our case I have used 0.1 as it gives better optimization results.

In [40]:
```
# Model Construct
model = ann_model(X_tf)

# Mean Squared Error cost function
cost = tf.reduce_mean(tf.square(y_tf - model))

# cost = tf.square(y_tf - model)
regularizer_1 = tf.nn.l2_loss(weights['w1'])
regularizer_2 = tf.nn.l2_loss(weights['w2'])
regularizer_3 = tf.nn.l2_loss(weights['w3'])
# cost = tf.reduce_mean(cost + beta1*regularizer_1 + beta2*regularizer_2 + beta3*regularizer_3)
cost = cost + beta1*regularizer_1 + beta2*regularizer_2 + beta3*regularizer_3

# Adam optimizer will update weights and biases after each step
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)


# Initialize variables
init = tf.global_variables_initializer()

# Add ops to save and restore all the variables.
saver = tf.train.Saver()
```

```
WARNING:tensorflow:From /home/navid/anaconda3/envs/tf/lib/python3.6/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int3
2 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
```

Training

In [ ]:
```
train_LC = []
val_LC = []
# session_var = None
```

Above train LC and val Lc variable keeps track of the learning rate so that learning curve can be drwan.

In [42]:
```
def training_block(X_train,y_train, X_test,y_test):
    #reseting variables
    session_var = None
    save_path = None

    with tf.Session() as sess:
        #running initializer
        sess.run(init)

#         minimum_validation_cost = 0.0190000
#         tf.Session.reset(sess)
        global minimum_validation_cost
        for i in range(num_steps):
    #         sess.run(optimizer, feed_dict={X_tf:X_train, y_tf:y_train})

            if submit :
                X_train = shuffle(train_processed , random_state = i)
                y_train = shuffle(target , random_state = i)
            else:
                X_train = shuffle(X_train , random_state = i)
                y_train = shuffle(y_train , random_state = i)

            trn_cost,_ = sess.run([cost,optimizer], feed_dict={X_tf:X_train, y_tf:y_train})
            tst_cost = sess.run(cost, feed_dict={X_tf:X_test, y_tf:y_test})
            if submit :
                new_minimum_validation_cost = np.min(trn_cost)
            else:
                new_minimum_validation_cost = np.min(tst_cost)

            if (i+1)%50 == 0:
                train_LC.append(trn_cost)
                val_LC.append(tst_cost)

            if (i+1)%500 == 0:
                print("epoch no : ",i+1, "  training cost: ",trn_cost, "  validation cost: ", tst_cost, "    minimum_validation_c
ost" , minimum_validation_cost)

            if new_minimum_validation_cost  < minimum_validation_cost :
                minimum_validation_cost = new_minimum_validation_cost
#                 global session_var
#                 session_var = sess
#                 Save the variables to disk.
                save_path = saver.save(sess, "model/model.ckpt")

        if bool(save_path):
            sess.close()
            print("Model saved in path: %s" % save_path)



training_block(X_train,y_train, X_test,y_test)
```

```
epoch no :  500   training cost:  4.48253   validation cost:  4.0299172     minimum_validation_cost 0.019
epoch no :  1000   training cost:  0.36521477   validation cost:  0.3668505     minimum_validation_cost 0.019
epoch no :  1500   training cost:  0.077625655   validation cost:  0.062232696     minimum_validation_cost 0.019
epoch no :  2000   training cost:  0.012304425   validation cost:  0.01430713     minimum_validation_cost 0.014317605
epoch no :  2500   training cost:  0.011156908   validation cost:  0.013151259     minimum_validation_cost 0.012850597
epoch no :  3000   training cost:  0.02404294   validation cost:  0.023661317     minimum_validation_cost 0.012850597
epoch no :  3500   training cost:  0.026674882   validation cost:  0.023832187     minimum_validation_cost 0.012850597
epoch no :  4000   training cost:  0.016393239   validation cost:  0.020106692     minimum_validation_cost 0.012850597
epoch no :  4500   training cost:  0.021710915   validation cost:  0.026027001     minimum_validation_cost 0.012850597
epoch no :  5000   training cost:  0.014663154   validation cost:  0.017669013     minimum_validation_cost 0.012850597
epoch no :  5500   training cost:  0.014944806   validation cost:  0.016238159     minimum_validation_cost 0.012850597
epoch no :  6000   training cost:  0.025703683   validation cost:  0.020001724     minimum_validation_cost 0.012850597
Model saved in path: model/model.ckpt
```

Grid search on epoch:

In the above block I have saved the model for the best validation score. As I mentioned earlier the epoch to reach the best validation accuracy is not fixed. Rather we can find it in 3 different range of epoch. The reason behind this is mostly because of random initializing of the weight and if we have fixed the seed value then it might change into only one single epoch range. But doing so we loose chance to improve our model further. Again if we want to ensemble different ANN model it woun't help when we use same seed and state. I have tried 1000+ parameters and combination from the start and used graph to visualize how to improve that but with grid search I might not get the exact idea why certain things provide good results or not and looking into every search result and graph is also too much so applying on the epoch seems to me more reasonable solution because the epoch for best validation result will be different in every run.

Trick

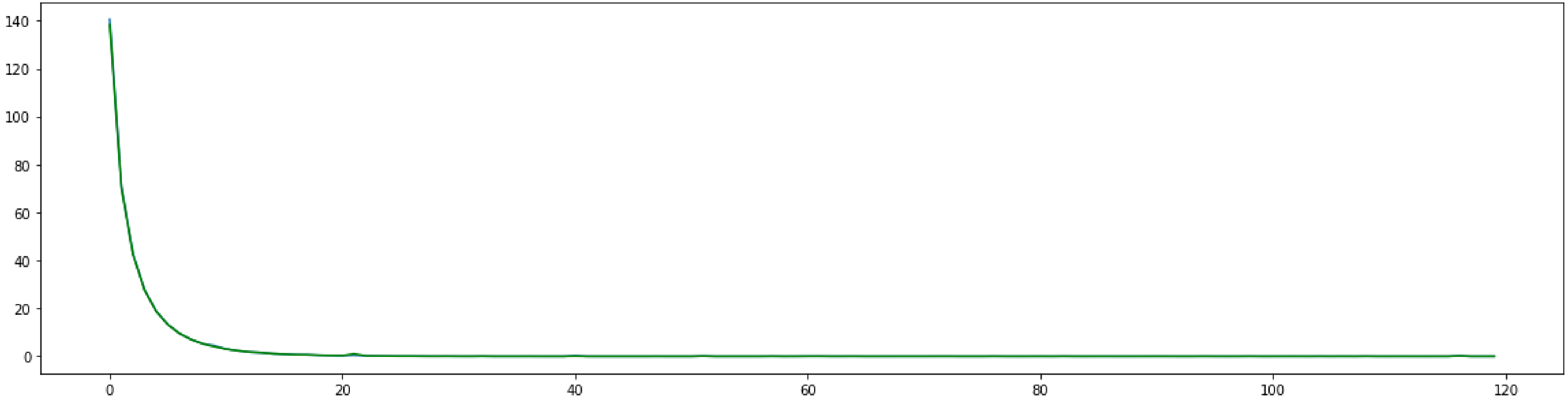
I have shuffled the data in every epoch and this trick improved the validation accuracy. On the other hand I did't use batch because according to my previous experience this kind of logistic regression problem works better when its given as a whole set rather than batch or mini-batch. But if its overfitting then passing the data in a batch / mini-batch would perform better as it helps to generalize more. We can say its more like a dropout effect. And I have tried to do dropout to reduce distance of training and validation accuracy but that didn't worked well.

In [43]:
```
def Prediction_block(X_test):
    with tf.Session() as sess:
        try:
            # Restore variables from disk.
            saver.restore(sess, "model/model.ckpt")
            print("Model restored.")
        except:
            print("------------ available checkpoint is for different model -------------")
            return
        # Check the values of the variables
        pred =  sess.run(model, feed_dict={X_tf: X_test})
        prediction = pred.squeeze()
        sess.close()
        return prediction
    #     print(np.exp(prediction))

prediction = Prediction_block(X_test)

pred_str = 'ANN_base_lr'+str(learning_rate)+'_beta'+str(beta1)+'-'+str(beta2)+'-'+str(beta3)+'-'+str(beta4)+'_hidden'+str(hidden_
1)+'-'+str(hidden_2)+'-'+str(hidden_3)+'-'+str(hidden_4)
prediction_dict[pred_str] = prediction

if submit:
    submit_prediction = Prediction_block(test_processed)
    submit_prediction_dict[pred_str] = submit_prediction
```

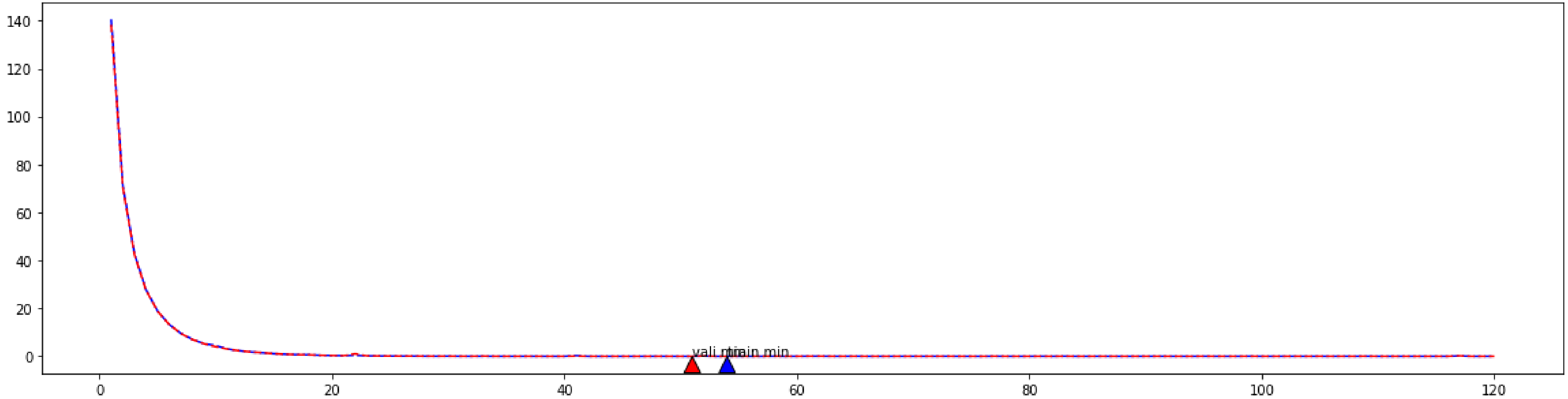
```
WARNING:tensorflow:From /home/navid/anaconda3/envs/tf/lib/python3.6/site-packages/tensorflow/python/training/saver.py:1266: check
point_exists (from tensorflow.python.training.checkpoint_management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
```

Learning curve

In [44]:
```
plt.figure(figsize=[20,5])
plt.plot(train_LC)
plt.plot(val_LC, 'g-')
plt.show()
```



In [45]:
```
def learning_curve():
    xdata = list(range(1,len(train_LC)+1))
    minimum = min(train_LC)

    plt.figure(figsize=[20,5])
    plt.plot(xdata, train_LC, 'b--')
    plt.annotate('train min', xy=(xdata[train_LC.index(minimum)], minimum),
                 arrowprops=dict(facecolor='blue', shrink=0.05))

    minimum = min(val_LC)
    plt.plot(xdata, val_LC, 'r--')
    plt.annotate('vali min', xy=(xdata[val_LC.index(minimum)], minimum),
                 arrowprops=dict(facecolor='red', shrink=0.05))

    plt.show()

learning_curve()
```



Acuracy Score

In [46]:
```
def accuracy(y_test,prediction):
    test_rmse_score = rmse(y_test, prediction)
    test_r2_score =  r2_score(np.array(y_test),prediction)
    return test_rmse_score, test_r2_score

test_rmse_score, test_r2_score = accuracy(y_test,prediction)

print('ann root mean absolute error: ', test_rmse_score)
print('accuracy score: ', test_r2_score  )
```

```
ann root mean absolute error:  0.10548200355447006
accuracy score:  0.9343473557962394
```

kaggle rmse:

In kaggle ranking the above ANN model provides the best rmse score and the score is 0.12324

Description on Learning curve and Accuracy:

We can observe where overfitting occurs. Overfitting actually occurs if the training loss goes under the validation loss even though the validation is still dropping. It is the sign that network is learning the patterns in the train set that are not applicable in the validation done. In a short note we can say::

Overfitting : training loss << validation loss

Underfitting : training loss >> validation loss

Just right : training loss ~ validation loss

According to this theory our both learning curve is exactly top of one another so in our case validation cost and training cost is almost same so we can say that our model is doing just the right thing. Again In validation score .1054 is impressive compared to other models.

Save score

In [130]:
```
if save_score:
    log_df = pd.read_csv("diffrent_training_results.csv")
    log_df = log_df.append({'learning_rate' : learning_rate, 'num_steps' : num_steps, 'beta1' : beta1, 'beta2' : beta2, 'beta3' : beta3, 'beta4' : beta4, 'hidden_1' : hidden_1 , 'hidden_2' : hidden_2, 'hidden_3' : hidden_3, 'hidden_4' : hidden_4, 'input_dim' : input_dim , 'test_rmse_score' : test_rmse_score , 'test_r2_score' : test_r2_score}, ignore_index=True)
    log_df.to_csv("diffrent_training_results.csv", encoding='utf-8',index=False)
```

Parameters

Following segment is actually initializing different parameters. From the dataset we can see that the estimation of sale price is a regression problem and neural network used here was overfitting most of the time due to higher variance. So for making it simpler I have penalized weight matrix of hidden layers with l2 regularization. Again I have found that single hidden layer with single neuron performs well and that means the prediction model don't need to be too complex. Thus I became ensured that regularization is going to improve performance.

Cross validation

When we perform a random train-test split of our data, we assume that our examples are independent. That means that by knowing/seeing some instance will not help us understand other instances. However, that’s not always the case. So to make sure if the Data is actually independent, to get more metrics and to use fine tuning my parameters on whole dataset I am performing cross validation.

In [49]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import RepeatedKFold
kf = KFold(n_splits=10, shuffle=True)

kf_rmse_list = []
kf_r2_list = []

# train_processed['SalePrice'] = target.values
for train_index, test_index in kf.split(train_processed):
    X_train, X_test = train_processed.iloc[train_index] , train_processed.iloc[test_index]
    y_train, y_test = target.iloc[train_index], target.iloc[test_index]

    training_block(X_train,y_train, X_test,y_test)
    prediction = Prediction_block(X_test)
    test_rmse_score, test_r2_score = accuracy(y_test, prediction)

    kf_rmse_list.append(test_rmse_score)
    kf_r2_list.append(test_r2_score)

    print("r2 list print", kf_r2_list)
    print('rmse list print',kf_rmse_list)

print("r2 mean print", np.mean(kf_r2_list))
print('rmse mean print', np.mean(kf_rmse_list))
```

```
epoch no :  500   training cost:  20.568653   validation cost:  20.612467     minimum_validation_cost 0.0139135355
epoch no :  1000   training cost:  4.473949   validation cost:  4.4875116     minimum_validation_cost 0.0139135355
epoch no :  1500   training cost:  1.1820256   validation cost:  1.1890986     minimum_validation_cost 0.0139135355
epoch no :  2000   training cost:  0.30349365   validation cost:  0.3106032     minimum_validation_cost 0.0139135355
epoch no :  2500   training cost:  0.07263037   validation cost:  0.080480844     minimum_validation_cost 0.0139135355
epoch no :  3000   training cost:  0.03073879   validation cost:  0.04442262     minimum_validation_cost 0.0139135355
epoch no :  3500   training cost:  0.014154274   validation cost:  0.02467569     minimum_validation_cost 0.0139135355
epoch no :  4000   training cost:  0.07976215   validation cost:  0.05189753     minimum_validation_cost 0.0139135355
epoch no :  4500   training cost:  0.039251808   validation cost:  0.04755907     minimum_validation_cost 0.0139135355
epoch no :  5000   training cost:  0.016424252   validation cost:  0.027453901     minimum_validation_cost 0.0139135355
epoch no :  5500   training cost:  0.012508737   validation cost:  0.022914646     minimum_validation_cost 0.0139135355
epoch no :  6000   training cost:  0.014900483   validation cost:  0.020737689     minimum_validation_cost 0.0139135355
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
r2 list print [0.9553614248860762]
rmse list print [0.09271964805908306]
epoch no :  500   training cost:  11.382257   validation cost:  11.393936     minimum_validation_cost 0.0139135355
epoch no :  1000   training cost:  1.2301323   validation cost:  1.2460473     minimum_validation_cost 0.0139135355
epoch no :  1500   training cost:  0.2234553   validation cost:  0.20101692     minimum_validation_cost 0.0139135355
epoch no :  2000   training cost:  0.042567566   validation cost:  0.053197816     minimum_validation_cost 0.0139135355
epoch no :  2500   training cost:  0.022891052   validation cost:  0.03237509     minimum_validation_cost 0.0139135355
epoch no :  3000   training cost:  0.02231348   validation cost:  0.036762033     minimum_validation_cost 0.0139135355
epoch no :  3500   training cost:  0.090252094   validation cost:  0.117978     minimum_validation_cost 0.0139135355
epoch no :  4000   training cost:  0.014654718   validation cost:  0.021233875     minimum_validation_cost 0.0139135355
epoch no :  4500   training cost:  0.021118347   validation cost:  0.030047739     minimum_validation_cost 0.0139135355
epoch no :  5000   training cost:  0.020689897   validation cost:  0.023326708     minimum_validation_cost 0.0139135355
epoch no :  5500   training cost:  0.055812918   validation cost:  0.037321463     minimum_validation_cost 0.0139135355
epoch no :  6000   training cost:  0.011782758   validation cost:  0.017926298     minimum_validation_cost 0.0139135355
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
r2 list print [0.9553614248860762, 0.9508771458271212]
rmse list print [0.09271964805908306, 0.0888583273296072]
epoch no :  500   training cost:  20.266645   validation cost:  20.1501     minimum_validation_cost 0.0139135355
epoch no :  1000   training cost:  4.9247584   validation cost:  4.872145     minimum_validation_cost 0.0139135355
epoch no :  1500   training cost:  1.7483015   validation cost:  1.7037398     minimum_validation_cost 0.0139135355
epoch no :  2000   training cost:  0.75342274   validation cost:  0.71101165     minimum_validation_cost 0.0139135355
epoch no :  2500   training cost:  0.3910143   validation cost:  0.34928468     minimum_validation_cost 0.0139135355
epoch no :  3000   training cost:  0.24408573   validation cost:  0.2026229     minimum_validation_cost 0.0139135355
epoch no :  3500   training cost:  0.18683618   validation cost:  0.14548528     minimum_validation_cost 0.0139135355
epoch no :  4000   training cost:  0.16872552   validation cost:  0.12734604     minimum_validation_cost 0.0139135355
epoch no :  4500   training cost:  0.16447207   validation cost:  0.12318214     minimum_validation_cost 0.0139135355
epoch no :  5000   training cost:  0.16388221   validation cost:  0.12260775     minimum_validation_cost 0.0139135355
epoch no :  5500   training cost:  0.16384023   validation cost:  0.12256751     minimum_validation_cost 0.0139135355
epoch no :  6000   training cost:  0.16384237   validation cost:  0.122562245     minimum_validation_cost 0.0139135355
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
r2 list print [0.9553614248860762, 0.9508771458271212, 0.9543415630626542]
rmse list print [0.09271964805908306, 0.0888583273296072, 0.07449732801900573]
epoch no :  500   training cost:  0.77447015   validation cost:  0.77047086     minimum_validation_cost 0.0139135355
epoch no :  1000   training cost:  0.023457717   validation cost:  0.024699744     minimum_validation_cost 0.0139135355
epoch no :  1500   training cost:  0.011655288   validation cost:  0.012237982     minimum_validation_cost 0.012241816
```

```
epoch no :  2000   training cost:  0.011911677   validation cost:  0.012319006     minimum_validation_cost 0.011949724
epoch no :  2500   training cost:  0.012986549   validation cost:  0.013175348     minimum_validation_cost 0.011949724
epoch no :  3000   training cost:  0.013242191   validation cost:  0.013814442     minimum_validation_cost 0.011949724
epoch no :  3500   training cost:  0.013824445   validation cost:  0.013548979     minimum_validation_cost 0.011949724
epoch no :  4000   training cost:  0.014795603   validation cost:  0.01416955     minimum_validation_cost 0.011949724
epoch no :  4500   training cost:  0.03727314   validation cost:  0.078026064     minimum_validation_cost 0.011949724
epoch no :  5000   training cost:  0.13976851   validation cost:  0.07447994     minimum_validation_cost 0.011949724
epoch no :  5500   training cost:  0.015592766   validation cost:  0.014788923     minimum_validation_cost 0.011949724
epoch no :  6000   training cost:  0.023306323   validation cost:  0.0315818     minimum_validation_cost 0.011949724
Model saved in path: model/model.ckpt
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
r2 list print [0.9553614248860762, 0.9508771458271212, 0.9543415630626542, 0.9341996754951252]
rmse list print [0.09271964805908306, 0.0888583273296072, 0.07449732801900573, 0.1007255124935721]
epoch no :  500   training cost:  4.137529   validation cost:  4.1315894     minimum_validation_cost 0.011949724
epoch no :  1000   training cost:  0.90885115   validation cost:  0.9184694     minimum_validation_cost 0.011949724
epoch no :  1500   training cost:  0.32985348   validation cost:  0.34109288     minimum_validation_cost 0.011949724
epoch no :  2000   training cost:  0.18882309   validation cost:  0.20048162     minimum_validation_cost 0.011949724
epoch no :  2500   training cost:  0.16195568   validation cost:  0.1737091     minimum_validation_cost 0.011949724
epoch no :  3000   training cost:  0.15870829   validation cost:  0.17048304     minimum_validation_cost 0.011949724
epoch no :  3500   training cost:  0.15851201   validation cost:  0.17027973     minimum_validation_cost 0.011949724
epoch no :  4000   training cost:  0.15850726   validation cost:  0.1702928     minimum_validation_cost 0.011949724
epoch no :  4500   training cost:  0.1585275   validation cost:  0.17028813     minimum_validation_cost 0.011949724
epoch no :  5000   training cost:  0.1585341   validation cost:  0.17028494     minimum_validation_cost 0.011949724
epoch no :  5500   training cost:  0.15850429   validation cost:  0.1702753     minimum_validation_cost 0.011949724
epoch no :  6000   training cost:  0.15850051   validation cost:  0.1702815     minimum_validation_cost 0.011949724
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
r2 list print [0.9553614248860762, 0.9508771458271212, 0.9543415630626542, 0.9341996754951252, 0.9504439704044774]
rmse list print [0.09271964805908306, 0.0888583273296072, 0.07449732801900573, 0.1007255124935721, 0.09159195884801004]
epoch no :  500   training cost:  0.18420246   validation cost:  0.17968395     minimum_validation_cost 0.011949724
epoch no :  1000   training cost:  0.16007997   validation cost:  0.15593112     minimum_validation_cost 0.011949724
epoch no :  1500   training cost:  0.160098   validation cost:  0.15596539     minimum_validation_cost 0.011949724
epoch no :  2000   training cost:  0.16007857   validation cost:  0.15593415     minimum_validation_cost 0.011949724
epoch no :  2500   training cost:  0.16015163   validation cost:  0.15594558     minimum_validation_cost 0.011949724
epoch no :  3000   training cost:  0.16008155   validation cost:  0.15593125     minimum_validation_cost 0.011949724
epoch no :  3500   training cost:  0.16009273   validation cost:  0.15595405     minimum_validation_cost 0.011949724
epoch no :  4000   training cost:  0.16012664   validation cost:  0.15593708     minimum_validation_cost 0.011949724
epoch no :  4500   training cost:  0.16007891   validation cost:  0.1559313     minimum_validation_cost 0.011949724
epoch no :  5000   training cost:  0.1601019   validation cost:  0.15595824     minimum_validation_cost 0.011949724
epoch no :  5500   training cost:  0.16012958   validation cost:  0.15597698     minimum_validation_cost 0.011949724
epoch no :  6000   training cost:  0.1601085   validation cost:  0.15594654     minimum_validation_cost 0.011949724
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
r2 list print [0.9553614248860762, 0.9508771458271212, 0.9543415630626542, 0.9341996754951252, 0.9504439704044774, 0.929886933228
908]
rmse list print [0.09271964805908306, 0.0888583273296072, 0.07449732801900573, 0.1007255124935721, 0.09159195884801004, 0.1045583
8190668743]
epoch no :  500   training cost:  0.17587255   validation cost:  0.19876517     minimum_validation_cost 0.011949724
epoch no :  1000   training cost:  0.15733822   validation cost:  0.18056849     minimum_validation_cost 0.011949724
epoch no :  1500   training cost:  0.157345   validation cost:  0.18057846     minimum_validation_cost 0.011949724
```

```
epoch no :  2000   training cost:  0.15734199   validation cost:  0.18057579     minimum_validation_cost 0.011949724
epoch no :  2500   training cost:  0.15734023   validation cost:  0.18057136     minimum_validation_cost 0.011949724
epoch no :  3000   training cost:  0.15734643   validation cost:  0.18057187     minimum_validation_cost 0.011949724
epoch no :  3500   training cost:  0.15733868   validation cost:  0.18056941     minimum_validation_cost 0.011949724
epoch no :  4000   training cost:  0.15737869   validation cost:  0.18057455     minimum_validation_cost 0.011949724
epoch no :  4500   training cost:  0.15742067   validation cost:  0.18062714     minimum_validation_cost 0.011949724
epoch no :  5000   training cost:  0.1573638   validation cost:  0.18058623     minimum_validation_cost 0.011949724
epoch no :  5500   training cost:  0.15738843   validation cost:  0.18061881     minimum_validation_cost 0.011949724
epoch no :  6000   training cost:  0.15735286   validation cost:  0.18060046     minimum_validation_cost 0.011949724
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
r2 list print [0.9553614248860762, 0.9508771458271212, 0.9543415630626542, 0.9341996754951252, 0.9504439704044774, 0.929886933228
908, 0.9382959161998816]
rmse list print [0.09271964805908306, 0.0888583273296072, 0.07449732801900573, 0.1007255124935721, 0.09159195884801004, 0.1045583
8190668743, 0.10551438239716186]
epoch no :  500   training cost:  0.19058843   validation cost:  0.19598773     minimum_validation_cost 0.011949724
epoch no :  1000   training cost:  0.12674819   validation cost:  0.090106644     minimum_validation_cost 0.011949724
epoch no :  1500   training cost:  0.010776039   validation cost:  0.015417905     minimum_validation_cost 0.011949724
epoch no :  2000   training cost:  0.011808951   validation cost:  0.016529392     minimum_validation_cost 0.011949724
epoch no :  2500   training cost:  0.012140089   validation cost:  0.017381266     minimum_validation_cost 0.011949724
epoch no :  3000   training cost:  0.020829326   validation cost:  0.01790555     minimum_validation_cost 0.011949724
epoch no :  3500   training cost:  0.01308694   validation cost:  0.016618628     minimum_validation_cost 0.011949724
epoch no :  4000   training cost:  0.017044349   validation cost:  0.021237336     minimum_validation_cost 0.011949724
epoch no :  4500   training cost:  0.013240088   validation cost:  0.016861184     minimum_validation_cost 0.011949724
epoch no :  5000   training cost:  0.013989   validation cost:  0.017500032     minimum_validation_cost 0.011949724
epoch no :  5500   training cost:  0.014921311   validation cost:  0.018324185     minimum_validation_cost 0.011949724
epoch no :  6000   training cost:  0.01547069   validation cost:  0.018815495     minimum_validation_cost 0.011949724
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
r2 list print [0.9553614248860762, 0.9508771458271212, 0.9543415630626542, 0.9341996754951252, 0.9504439704044774, 0.929886933228
908, 0.9382959161998816, 0.932418870844409]
rmse list print [0.09271964805908306, 0.0888583273296072, 0.07449732801900573, 0.1007255124935721, 0.09159195884801004, 0.1045583
8190668743, 0.10551438239716186, 0.10189308411814846]
epoch no :  500   training cost:  14.345135   validation cost:  14.309958     minimum_validation_cost 0.011949724
epoch no :  1000   training cost:  1.9079558   validation cost:  1.9146067     minimum_validation_cost 0.011949724
epoch no :  1500   training cost:  0.45495993   validation cost:  0.37160826     minimum_validation_cost 0.011949724
epoch no :  2000   training cost:  0.10315405   validation cost:  0.10432874     minimum_validation_cost 0.011949724
epoch no :  2500   training cost:  0.048060406   validation cost:  0.07186215     minimum_validation_cost 0.011949724
epoch no :  3000   training cost:  0.022855159   validation cost:  0.021485604     minimum_validation_cost 0.011949724
epoch no :  3500   training cost:  0.012944125   validation cost:  0.012365152     minimum_validation_cost 0.010917494
epoch no :  4000   training cost:  0.011641011   validation cost:  0.010896233     minimum_validation_cost 0.010901678
epoch no :  4500   training cost:  0.011573468   validation cost:  0.0106407385     minimum_validation_cost 0.010396154
epoch no :  5000   training cost:  0.012472791   validation cost:  0.011355851     minimum_validation_cost 0.010396154
epoch no :  5500   training cost:  0.012279389   validation cost:  0.011330111     minimum_validation_cost 0.010396154
epoch no :  6000   training cost:  0.0148251755   validation cost:  0.01329639     minimum_validation_cost 0.010396154
Model saved in path: model/model.ckpt
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
r2 list print [0.9553614248860762, 0.9508771458271212, 0.9543415630626542, 0.9341996754951252, 0.9504439704044774, 0.929886933228
908, 0.9382959161998816, 0.932418870844409, 0.9366822053541947]
rmse list print [0.09271964805908306, 0.0888583273296072, 0.07449732801900573, 0.1007255124935721, 0.09159195884801004, 0.1045583
```

```
8190668743, 0.10551438239716186, 0.10189308411814846, 0.09402134306063017]
epoch no :  500   training cost:  2.672395   validation cost:  2.670042     minimum_validation_cost 0.010396154
epoch no :  1000   training cost:  0.13111192   validation cost:  0.13321456     minimum_validation_cost 0.010396154
epoch no :  1500   training cost:  0.011619842   validation cost:  0.019234128     minimum_validation_cost 0.010396154
epoch no :  2000   training cost:  0.009893612   validation cost:  0.017380662     minimum_validation_cost 0.010396154
epoch no :  2500   training cost:  0.010010591   validation cost:  0.017384036     minimum_validation_cost 0.010396154
epoch no :  3000   training cost:  0.015444754   validation cost:  0.02303415     minimum_validation_cost 0.010396154
epoch no :  3500   training cost:  0.015875159   validation cost:  0.028511856     minimum_validation_cost 0.010396154
epoch no :  4000   training cost:  0.012220575   validation cost:  0.018918835     minimum_validation_cost 0.010396154
epoch no :  4500   training cost:  0.013949025   validation cost:  0.022416791     minimum_validation_cost 0.010396154
epoch no :  5000   training cost:  0.013269598   validation cost:  0.020984132     minimum_validation_cost 0.010396154
epoch no :  5500   training cost:  0.013208384   validation cost:  0.019760473     minimum_validation_cost 0.010396154
epoch no :  6000   training cost:  0.026704477   validation cost:  0.031736426     minimum_validation_cost 0.010396154
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
r2 list print [0.9553614248860762, 0.9508771458271212, 0.9543415630626542, 0.9341996754951252, 0.9504439704044774, 0.929886933228
908, 0.9382959161998816, 0.932418870844409, 0.9366822053541947, 0.9255999030483704]
rmse list print [0.09271964805908306, 0.0888583273296072, 0.07449732801900573, 0.1007255124935721, 0.09159195884801004, 0.1045583
8190668743, 0.10551438239716186, 0.10189308411814846, 0.09402134306063017, 0.11009827566438594]
r2 mean print 0.9408107608351217
rmse mean print 0.09644782418962919
```

Observation

In the cross validation section we can see that 10 fold cross validation on our best ANN model provides similar rmse to 80-20 split rmse score. So we can relay on 80-20 split on this dataset. Thus we can say that the data in the dataset is independent.

Observing Few Other well performed ANN models

In this section We are observing the few other models and their learning curve. After that some of them will be used for Ensemble learning section for further improvement. In this model I have only changed the size of hidden layer, amount of neuron in each hidden layers , number of steps and learning rates. Rest of the part is same as the ANN described above.

ANN with 4 layers

Initialization of models

In [100]:
```
tf.reset_default_graph()
def weight_bais():
    global weights, biases
    weights = None
    biases = None

    weights = {
        'w1': tf.Variable(tf.random_normal([input_dim, hidden_1])),
        'w2': tf.Variable(tf.random_normal([hidden_1, hidden_2])),
        'w3': tf.Variable(tf.random_normal([hidden_2, hidden_3])),
        'w4': tf.Variable(tf.random_normal([hidden_3, hidden_4])),
        'out': tf.Variable(tf.random_normal([hidden_4, output_dim]))
    }
    biases = {
        'b1': tf.Variable(tf.random_normal([hidden_1])),
        'b2': tf.Variable(tf.random_normal([hidden_2])),
        'b3': tf.Variable(tf.random_normal([hidden_3])),
        'b4': tf.Variable(tf.random_normal([hidden_4])),
        'out': tf.Variable(tf.random_normal([output_dim]))
    }
```

In [101]:
```
def ann_model(X_val):
  # Hidden layers
    layer_1 = tf.add(tf.matmul(X_val, weights['w1']), biases['b1'])
    layer_1 = tf.nn.relu(layer_1)


    layer_2 = tf.add(tf.matmul(layer_1, weights['w2']), biases['b2'])
    layer_2 = tf.nn.relu(layer_2)

    layer_3 = tf.add(tf.matmul(layer_2, weights['w3']), biases['b3'])
    layer_3 = tf.nn.relu(layer_3)

    layer_4 = tf.add(tf.matmul(layer_3, weights['w4']), biases['b4'])
    layer_4 = tf.nn.relu(layer_4)

    # Output layer
    layer_out = tf.add(tf.matmul(layer_4, weights['out']), biases['out'])

    return layer_out
```

In [102]:
```
regularizer_4 = None
def miscellaneous_initialization():
    global model, cost , regularizer_1 , regularizer_2 ,regularizer_3, regularizer_4, optimizer , init , saver
    # Model Construct
    model = ann_model(X_tf)

    # Mean Squared Error cost function
    cost = tf.reduce_mean(tf.square(y_tf - model))

    # cost = tf.square(y_tf - model)
    regularizer_1 = tf.nn.l2_loss(weights['w1'])
    regularizer_2 = tf.nn.l2_loss(weights['w2'])
    regularizer_3 = tf.nn.l2_loss(weights['w3'])
    regularizer_4 = tf.nn.l2_loss(weights['w4'])
    # cost = tf.reduce_mean(cost + beta1*regularizer_1 + beta2*regularizer_2 + beta3*regularizer_3)
    cost = cost + beta1*regularizer_1 + beta2*regularizer_2 + beta3*regularizer_3 + beta4*regularizer_4

    # Adam optimizer will update weights and biases after each step
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)


    # Initialize variables
    init = tf.global_variables_initializer()

    # Add ops to save and restore all the variables.
    saver = tf.train.Saver()
```

Training

ANN 1

In this section changed variables are

- learning rate = .01

layer name	Neuron	value of beta for l2 regularization
1st hidden layer	76 Neuron	.1
2nd hidden layer	48 Neuron	.05
3rd hidden layer	32 Neuron	0
4th hidden layer	16 Neuron	0

In [253]:

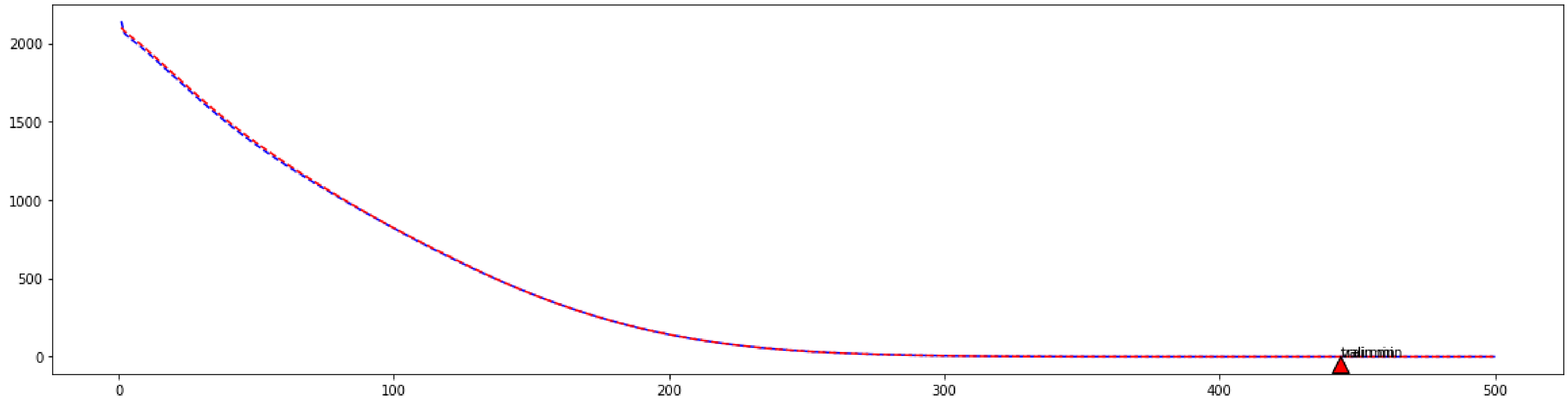
```
tf.reset_default_graph()
learning_rate = 0.1
num_steps = 25000
#for regularize weight matrix
beta1 = 0.1
beta2 = 0.05
beta3 = 0.00
beta4 = 0.0

hidden_1 = 76
hidden_2 = 48
hidden_3 = 32
hidden_4 = 16

minimum_validation_cost = .02101000

input_dim = X_train.shape[1]  # Number of features
output_dim = 1               # Because it is a regression problem

#tf graph input
X_tf = tf.placeholder("float" )
y_tf = tf.placeholder("float" )

weight_bais()
miscellaneous_initialization()
train_LC = []
val_LC = []
```

In [254]:

```
training_block(X_train,y_train, X_test,y_test)
prediction = Prediction_block(X_test)
test_rmse_score, test_r2_score = accuracy(y_test,prediction)

print('ann root mean absolute error: ', test_rmse_score)
print('accuracy score: ', test_r2_score  )
learning_curve()

pred_str = 'ANN_lr'+str(learning_rate)+'_beta'+str(beta1)+'-'+str(beta2)+'-'+str(beta3)+'-'+str(beta4)+'_hidden'+str(hidden_1)+'-'+str(hidden_2)+'-'+str(hidden_3)+'-'+str(hidden_4)
prediction_dict[pred_str] = prediction

if submit:
    submit_prediction = Prediction_block(test_processed)
    submit_prediction_dict[pred_str] = submit_prediction
# Data Save
if save_score:
    log_df = pd.read_csv("diffrent_training_results.csv")
    log_df = log_df.append({'learning_rate' : learning_rate, 'num_steps' : num_steps, 'beta1' : beta1, 'beta2' : beta2, 'beta3' : beta3, 'beta4' : beta4, 'hidden_1' : hidden_1 , 'hidden_2' : hidden_2, 'hidden_3' : hidden_3, 'hidden_4' : hidden_4, 'input_dim' : input_dim , 'test_rmse_score' : test_rmse_score , 'test_r2_score' : test_r2_score}, ignore_index=True)
    log_df.to_csv("diffrent_training_results.csv", encoding='utf-8',index=False)
```

```
epoch no :  500   training cost:  1948.0289   validation cost:  1963.5165     minimum_validation_cost 0.02101
epoch no :  1000   training cost:  1789.3033   validation cost:  1804.5315     minimum_validation_cost 0.02101
epoch no :  1500   training cost:  1631.0383   validation cost:  1646.2167     minimum_validation_cost 0.02101
epoch no :  2000   training cost:  1484.1322   validation cost:  1499.1335     minimum_validation_cost 0.02101
epoch no :  2500   training cost:  1351.5776   validation cost:  1365.8958     minimum_validation_cost 0.02101
epoch no :  3000   training cost:  1231.5188   validation cost:  1244.1323     minimum_validation_cost 0.02101
epoch no :  3500   training cost:  1121.0094   validation cost:  1129.8462     minimum_validation_cost 0.02101
epoch no :  4000   training cost:  1016.4562   validation cost:  1022.328     minimum_validation_cost 0.02101
epoch no :  4500   training cost:  915.9035   validation cost:  919.5465     minimum_validation_cost 0.02101
epoch no :  5000   training cost:  818.86554   validation cost:  821.1494     minimum_validation_cost 0.02101
epoch no :  5500   training cost:  725.55963   validation cost:  729.7903     minimum_validation_cost 0.02101
epoch no :  6000   training cost:  636.4901   validation cost:  640.9389     minimum_validation_cost 0.02101
epoch no :  6500   training cost:  552.2778   validation cost:  555.0504     minimum_validation_cost 0.02101
epoch no :  7000   training cost:  473.5905   validation cost:  474.14694     minimum_validation_cost 0.02101
epoch no :  7500   training cost:  401.07443   validation cost:  401.0503     minimum_validation_cost 0.02101
epoch no :  8000   training cost:  335.22263   validation cost:  335.2226     minimum_validation_cost 0.02101
epoch no :  8500   training cost:  276.37018   validation cost:  276.4013     minimum_validation_cost 0.02101
epoch no :  9000   training cost:  224.65007   validation cost:  224.68013     minimum_validation_cost 0.02101
epoch no :  9500   training cost:  179.96939   validation cost:  179.98999     minimum_validation_cost 0.02101
epoch no :  10000   training cost:  142.05312   validation cost:  142.03314     minimum_validation_cost 0.02101
epoch no :  10500   training cost:  110.439835   validation cost:  110.40212     minimum_validation_cost 0.02101
epoch no :  11000   training cost:  84.54955   validation cost:  84.55469     minimum_validation_cost 0.02101
epoch no :  11500   training cost:  63.74702   validation cost:  63.77253     minimum_validation_cost 0.02101
epoch no :  12000   training cost:  47.36451   validation cost:  47.38588     minimum_validation_cost 0.02101
epoch no :  12500   training cost:  34.71665   validation cost:  34.722527     minimum_validation_cost 0.02101
epoch no :  13000   training cost:  25.131432   validation cost:  25.155582     minimum_validation_cost 0.02101
epoch no :  13500   training cost:  17.955591   validation cost:  17.975864     minimum_validation_cost 0.02101
epoch no :  14000   training cost:  12.609146   validation cost:  12.616015     minimum_validation_cost 0.02101
epoch no :  14500   training cost:  8.702005   validation cost:  8.7001095     minimum_validation_cost 0.02101
epoch no :  15000   training cost:  5.90036   validation cost:  5.8999863     minimum_validation_cost 0.02101
epoch no :  15500   training cost:  3.8992198   validation cost:  3.8999383     minimum_validation_cost 0.02101
epoch no :  16000   training cost:  2.5155318   validation cost:  2.5166104     minimum_validation_cost 0.02101
epoch no :  16500   training cost:  1.5840064   validation cost:  1.585098     minimum_validation_cost 0.02101
epoch no :  17000   training cost:  0.9745886   validation cost:  0.9757318     minimum_validation_cost 0.02101
epoch no :  17500   training cost:  0.5837551   validation cost:  0.5853919     minimum_validation_cost 0.02101
epoch no :  18000   training cost:  0.34044278   validation cost:  0.34229845     minimum_validation_cost 0.02101
epoch no :  18500   training cost:  0.19107433   validation cost:  0.19231078     minimum_validation_cost 0.02101
epoch no :  19000   training cost:  0.106256224   validation cost:  0.10772059     minimum_validation_cost 0.02101
epoch no :  19500   training cost:  0.060021322   validation cost:  0.062279984     minimum_validation_cost 0.02101
epoch no :  20000   training cost:  0.034057993   validation cost:  0.03559231     minimum_validation_cost 0.02101
epoch no :  20500   training cost:  0.022776678   validation cost:  0.024662416     minimum_validation_cost 0.02101
epoch no :  21000   training cost:  0.026222728   validation cost:  0.026665546     minimum_validation_cost 0.019980568
epoch no :  21500   training cost:  0.018489484   validation cost:  0.01983606     minimum_validation_cost 0.018412706
epoch no :  22000   training cost:  0.01716589   validation cost:  0.018984884     minimum_validation_cost 0.018258983
epoch no :  22500   training cost:  0.019204231   validation cost:  0.020943124     minimum_validation_cost 0.018142112
epoch no :  23000   training cost:  0.023870615   validation cost:  0.031735398     minimum_validation_cost 0.018142112
epoch no :  23500   training cost:  0.017715417   validation cost:  0.022381995     minimum_validation_cost 0.018142112
epoch no :  24000   training cost:  0.019991433   validation cost:  0.023535162     minimum_validation_cost 0.018142112
epoch no :  24500   training cost:  0.02650885   validation cost:  0.025393497     minimum_validation_cost 0.018142112
epoch no :  25000   training cost:  0.061272897   validation cost:  0.2311845     minimum_validation_cost 0.018142112
Model saved in path: model/model.ckpt
```

```
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
ann root mean absolute error:  0.11033837928515242
accuracy score:  0.9281629180762654
```



ANN 2

In this section changed variables are

- learning rate = .05

layer name	Neuron	value of beta for l2 regularization
1st hidden layer	8 Neuron	.005
2nd hidden layer	32 Neuron	.1
3rd hidden layer	16 Neuron	0.05
4th hidden layer	8 Neuron	0

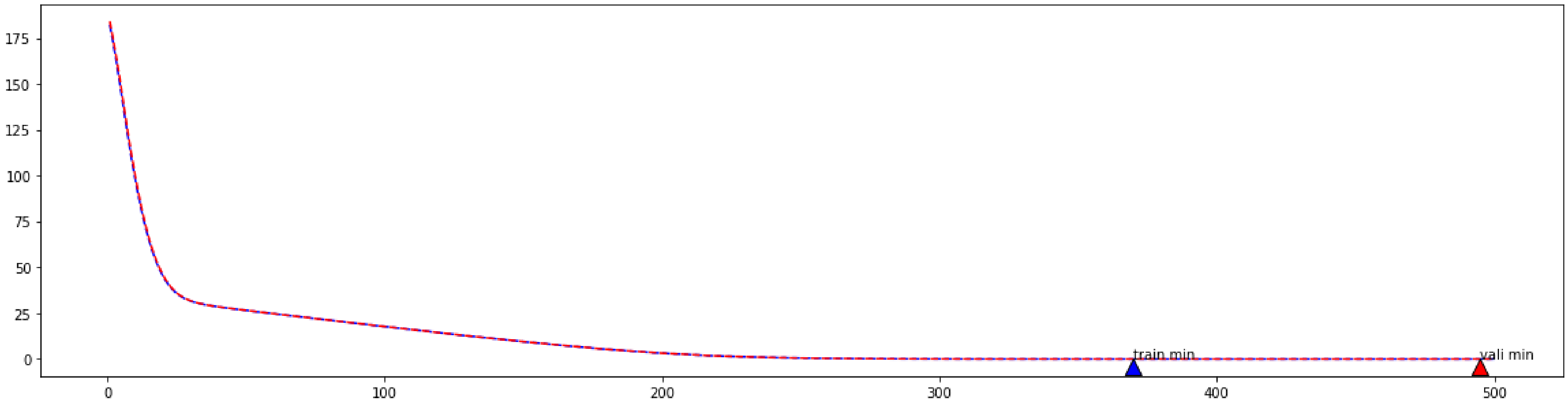
In [55]:

```
tf.reset_default_graph()
learning_rate = 0.05
num_steps = 25000
#for regularize weight matrix
beta1 = 0.005
beta2 = 0.1
beta3 = 0.05
beta4 = 0.0

hidden_1 = 8
hidden_2 = 32
hidden_3 = 16
hidden_4 = 8


minimum_validation_cost = 0.02101000
#tf graph input
X_tf = tf.placeholder("float" )
y_tf = tf.placeholder("float" )

weight_bais()
miscellaneous_initialization()
train_LC = []
val_LC = []
```

In [56]:
```
training_block(X_train,y_train, X_test,y_test)
prediction = Prediction_block(X_test)

test_rmse_score, test_r2_score = accuracy(y_test,prediction)

print('ann root mean absolute error: ', test_rmse_score)
print('accuracy score: ', test_r2_score  )
learning_curve()



pred_str = 'ANN_lr'+str(learning_rate)+'_beta'+str(beta1)+'-'+str(beta2)+'-'+str(beta3)+'-'+str(beta4)+'_hidden'+str(hidden_1)+'-
'+str(hidden_2)+'-'+str(hidden_3)+'-'+str(hidden_4)
prediction_dict[pred_str] = prediction

if submit:
    submit_prediction = Prediction_block(test_processed)
    submit_prediction_dict[pred_str] = submit_prediction
# Data Save
if save_score:
    log_df = pd.read_csv("diffrent_training_results.csv")
    log_df = log_df.append({'learning_rate' : learning_rate, 'num_steps' : num_steps, 'beta1' : beta1, 'beta2' : beta2, 'beta3' :
beta3, 'beta4' : beta4, 'hidden_1' : hidden_1 , 'hidden_2' : hidden_2, 'hidden_3' : hidden_3, 'hidden_4' : hidden_4, 'input_dim'
: input_dim , 'test_rmse_score' : test_rmse_score , 'test_r2_score' : test_r2_score}, ignore_index=True)
    log_df.to_csv("diffrent_training_results.csv", encoding='utf-8',index=False)
```

```
epoch no :  500   training cost:  98.96163   validation cost:  101.16655    minimum_validation_cost 0.02101
epoch no :  1000   training cost:  45.49598   validation cost:  46.24627    minimum_validation_cost 0.02101
epoch no :  1500   training cost:  31.685268   validation cost:  31.812305    minimum_validation_cost 0.02101
epoch no :  2000   training cost:  28.494917   validation cost:  28.509579    minimum_validation_cost 0.02101
epoch no :  2500   training cost:  26.541996   validation cost:  26.54794    minimum_validation_cost 0.02101
epoch no :  3000   training cost:  24.718468   validation cost:  24.729013    minimum_validation_cost 0.02101
epoch no :  3500   training cost:  22.942047   validation cost:  22.95431    minimum_validation_cost 0.02101
epoch no :  4000   training cost:  21.18252   validation cost:  21.19542    minimum_validation_cost 0.02101
epoch no :  4500   training cost:  19.437643   validation cost:  19.448608    minimum_validation_cost 0.02101
epoch no :  5000   training cost:  17.709394   validation cost:  17.720444    minimum_validation_cost 0.02101
epoch no :  5500   training cost:  15.999163   validation cost:  16.01024    minimum_validation_cost 0.02101
epoch no :  6000   training cost:  14.31156   validation cost:  14.322832    minimum_validation_cost 0.02101
epoch no :  6500   training cost:  12.659315   validation cost:  12.677395    minimum_validation_cost 0.02101
epoch no :  7000   training cost:  11.062086   validation cost:  11.081362    minimum_validation_cost 0.02101
epoch no :  7500   training cost:  9.52519   validation cost:  9.536628    minimum_validation_cost 0.02101
epoch no :  8000   training cost:  8.041063   validation cost:  8.0620365    minimum_validation_cost 0.02101
epoch no :  8500   training cost:  6.627181   validation cost:  6.6679873    minimum_validation_cost 0.02101
epoch no :  9000   training cost:  5.356043   validation cost:  5.36625    minimum_validation_cost 0.02101
epoch no :  9500   training cost:  4.2130976   validation cost:  4.231335    minimum_validation_cost 0.02101
epoch no :  10000   training cost:  3.2202468   validation cost:  3.2339666    minimum_validation_cost 0.02101
epoch no :  10500   training cost:  2.4022565   validation cost:  2.420408    minimum_validation_cost 0.02101
epoch no :  11000   training cost:  1.7369599   validation cost:  1.7432039    minimum_validation_cost 0.02101
epoch no :  11500   training cost:  1.2229878   validation cost:  1.2248117    minimum_validation_cost 0.02101
epoch no :  12000   training cost:  0.8650217   validation cost:  0.8522291    minimum_validation_cost 0.02101
epoch no :  12500   training cost:  0.5642521   validation cost:  0.5668021    minimum_validation_cost 0.02101
epoch no :  13000   training cost:  0.3693418   validation cost:  0.37263116    minimum_validation_cost 0.02101
epoch no :  13500   training cost:  0.23565769   validation cost:  0.23976843    minimum_validation_cost 0.02101
epoch no :  14000   training cost:  0.15317412   validation cost:  0.15297893    minimum_validation_cost 0.02101
epoch no :  14500   training cost:  0.089964926   validation cost:  0.095126145    minimum_validation_cost 0.02101
epoch no :  15000   training cost:  0.05444552   validation cost:  0.060352698    minimum_validation_cost 0.02101
epoch no :  15500   training cost:  0.033405006   validation cost:  0.04072603    minimum_validation_cost 0.02101
epoch no :  16000   training cost:  0.021379678   validation cost:  0.02719137    minimum_validation_cost 0.02101
epoch no :  16500   training cost:  0.017818375   validation cost:  0.02784073    minimum_validation_cost 0.02101
epoch no :  17000   training cost:  0.01326775   validation cost:  0.021916628    minimum_validation_cost 0.020162757
epoch no :  17500   training cost:  0.017692542   validation cost:  0.026436877    minimum_validation_cost 0.018723719
epoch no :  18000   training cost:  0.012610511   validation cost:  0.019203333    minimum_validation_cost 0.018243955
epoch no :  18500   training cost:  0.011309475   validation cost:  0.018362269    minimum_validation_cost 0.018199008
epoch no :  19000   training cost:  0.017165741   validation cost:  0.026090968    minimum_validation_cost 0.018195953
epoch no :  19500   training cost:  0.017472688   validation cost:  0.025563277    minimum_validation_cost 0.018068379
epoch no :  20000   training cost:  0.01170708   validation cost:  0.018411675    minimum_validation_cost 0.017956132
epoch no :  20500   training cost:  0.017795723   validation cost:  0.023501392    minimum_validation_cost 0.017956132
epoch no :  21000   training cost:  0.013715736   validation cost:  0.02364869    minimum_validation_cost 0.017956132
epoch no :  21500   training cost:  0.01588199   validation cost:  0.024066374    minimum_validation_cost 0.01679543
epoch no :  22000   training cost:  0.013990714   validation cost:  0.019897664    minimum_validation_cost 0.01679543
epoch no :  22500   training cost:  0.026997136   validation cost:  0.021226006    minimum_validation_cost 0.01679543
epoch no :  23000   training cost:  0.014782673   validation cost:  0.019962754    minimum_validation_cost 0.01679543
epoch no :  23500   training cost:  0.02022815   validation cost:  0.026848074    minimum_validation_cost 0.01679543
epoch no :  24000   training cost:  0.016962774   validation cost:  0.021317616    minimum_validation_cost 0.01679543
epoch no :  24500   training cost:  0.024479048   validation cost:  0.02893193    minimum_validation_cost 0.01679543
epoch no :  25000   training cost:  0.021377431   validation cost:  0.033468053    minimum_validation_cost 0.01679543
Model saved in path: model/model.ckpt
```

```
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
ann root mean absolute error:  0.10818300213571848
accuracy score:  0.9309420725912334
```



ANN 3

- learning rate = .05

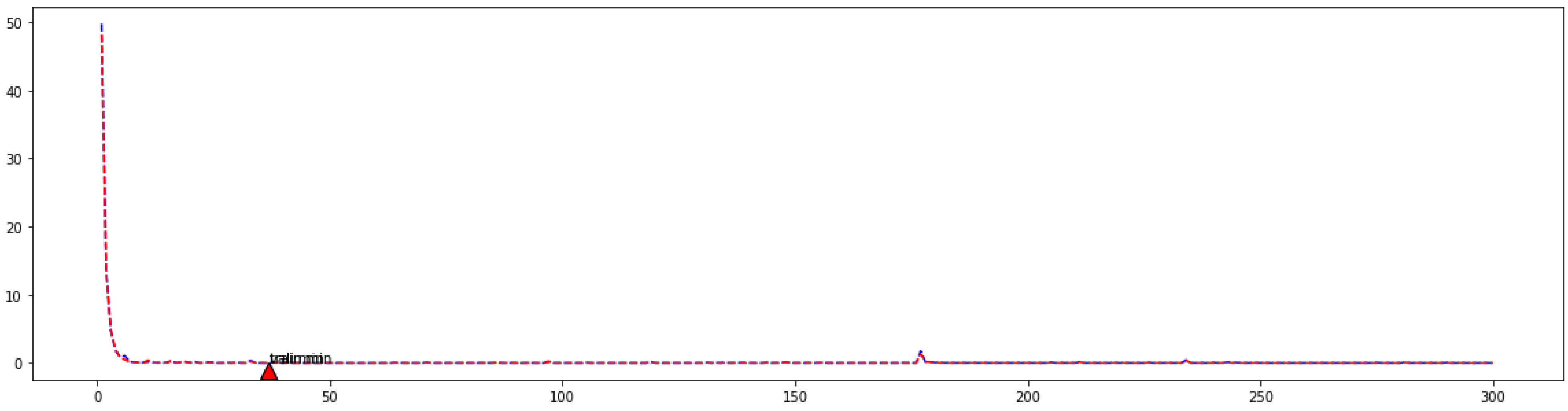
layer name	Neuron	value of beta for l2 regularization
1st hidden layer	16 Neuron	.1
2nd hidden layer	8 Neuron	.0
3rd hidden layer	4 Neuron	0.0
4th hidden layer	2 Neuron	0

In [63]:
```
tf.reset_default_graph()
learning_rate = 0.05
num_steps = 15000
#for regularize weight matrix
beta1 = 0.1
beta2 = 0.0
beta3 = 0.00
beta4 = 0.0

hidden_1 = 16
hidden_2 = 8
hidden_3 = 4
hidden_4 = 2

minimum_validation_cost = 0.01901000

#tf graph input
X_tf = tf.placeholder("float" )
y_tf = tf.placeholder("float" )


weight_bais()
miscellaneous_initialization()
train_LC = []
val_LC = []
```

In [64]:

```
training_block(X_train,y_train, X_test,y_test)
prediction = Prediction_block(X_test)
test_rmse_score, test_r2_score = accuracy(y_test,prediction)

print('ann root mean absolute error: ', test_rmse_score)
print('accuracy score: ', test_r2_score  )
learning_curve()

pred_str = 'ANN_lr'+str(learning_rate)+'_beta'+str(beta1)+'-'+str(beta2)+'-'+str(beta3)+'-'+str(beta4)+'_hidden'+str(hidden_1)+'-'+str(hidden_2)+'-'+str(hidden_3)+'-'+str(hidden_4)
prediction_dict[pred_str] = prediction

if submit:
    submit_prediction = Prediction_block(test_processed)
    submit_prediction_dict[pred_str] = submit_prediction
# Data Save
if save_score:
    log_df = pd.read_csv("diffrent_training_results.csv")
    log_df = log_df.append({'learning_rate' : learning_rate, 'num_steps' : num_steps, 'beta1' : beta1, 'beta2' : beta2, 'beta3' : beta3, 'beta4' : beta4, 'hidden_1' : hidden_1 , 'hidden_2' : hidden_2, 'hidden_3' : hidden_3, 'hidden_4' : hidden_4, 'input_dim' : input_dim , 'test_rmse_score' : test_rmse_score , 'test_r2_score' : test_r2_score}, ignore_index=True)
    log_df.to_csv("diffrent_training_results.csv", encoding='utf-8',index=False)
```

```
epoch no :  500   training cost:  0.06750769   validation cost:  0.07170834    minimum_validation_cost 0.01901
epoch no :  1000   training cost:  0.024438005   validation cost:  0.027701719    minimum_validation_cost 0.01901
epoch no :  1500   training cost:  0.10088402   validation cost:  0.03532496    minimum_validation_cost 0.015040747
epoch no :  2000   training cost:  0.011049605   validation cost:  0.013185406    minimum_validation_cost 0.012483893
epoch no :  2500   training cost:  0.011799688   validation cost:  0.013729342    minimum_validation_cost 0.012483893
epoch no :  3000   training cost:  0.013513778   validation cost:  0.014625475    minimum_validation_cost 0.012483893
epoch no :  3500   training cost:  0.01181775   validation cost:  0.013272421    minimum_validation_cost 0.012483893
epoch no :  4000   training cost:  0.013972165   validation cost:  0.015514897    minimum_validation_cost 0.012483893
epoch no :  4500   training cost:  0.012848759   validation cost:  0.014008677    minimum_validation_cost 0.012483893
epoch no :  5000   training cost:  0.01293095   validation cost:  0.014150724    minimum_validation_cost 0.012483893
epoch no :  5500   training cost:  0.015687615   validation cost:  0.014634018    minimum_validation_cost 0.012483893
epoch no :  6000   training cost:  0.013886566   validation cost:  0.015381509    minimum_validation_cost 0.012483893
epoch no :  6500   training cost:  0.016043557   validation cost:  0.016748872    minimum_validation_cost 0.012483893
epoch no :  7000   training cost:  0.014979528   validation cost:  0.016671315    minimum_validation_cost 0.012483893
epoch no :  7500   training cost:  0.016729917   validation cost:  0.019415256    minimum_validation_cost 0.012483893
epoch no :  8000   training cost:  0.0152445175   validation cost:  0.016355077    minimum_validation_cost 0.012483893
epoch no :  8500   training cost:  0.015557777   validation cost:  0.016310645    minimum_validation_cost 0.012483893
epoch no :  9000   training cost:  0.07586811   validation cost:  0.07899709    minimum_validation_cost 0.012483893
epoch no :  9500   training cost:  0.02142299   validation cost:  0.024720391    minimum_validation_cost 0.012483893
epoch no :  10000   training cost:  0.024426803   validation cost:  0.027593074    minimum_validation_cost 0.012483893
epoch no :  10500   training cost:  0.018516378   validation cost:  0.019000562    minimum_validation_cost 0.012483893
epoch no :  11000   training cost:  0.048320018   validation cost:  0.048580103    minimum_validation_cost 0.012483893
epoch no :  11500   training cost:  0.020851415   validation cost:  0.037529126    minimum_validation_cost 0.012483893
epoch no :  12000   training cost:  0.058065794   validation cost:  0.029545134    minimum_validation_cost 0.012483893
epoch no :  12500   training cost:  0.015809786   validation cost:  0.016998455    minimum_validation_cost 0.012483893
epoch no :  13000   training cost:  0.01982437   validation cost:  0.02025333    minimum_validation_cost 0.012483893
epoch no :  13500   training cost:  0.014694117   validation cost:  0.016001858    minimum_validation_cost 0.012483893
epoch no :  14000   training cost:  0.0147103565   validation cost:  0.015879849    minimum_validation_cost 0.012483893
epoch no :  14500   training cost:  0.071125045   validation cost:  0.035555735    minimum_validation_cost 0.012483893
epoch no :  15000   training cost:  0.023572352   validation cost:  0.016225783    minimum_validation_cost 0.012483893
Model saved in path: model/model.ckpt
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
ann root mean absolute error:  0.10504402603717251
accuracy score:  0.9348914236909488
```



Description on Learning curve and Accuracy:

We can observe where overfitting occurs. Overfitting actually occurs if the training loss goes under the validation loss even though the validation is still dropping. It is the sign that network is learning the patterns in the train set that are not applicable in the validation done. In a short note we can say::

Overfitting : training loss << validation loss

Underfitting : training loss >> validation loss

Just right : training loss ~ validation loss

According to this theory, for ANN 1,2 and 3 our both learning curve (validation cost and training cost) is exactly top of one another so in our case validation cost and training cost is almost same so we can say that our model is doing just the right thing. Again In validation score .11,.1081 and .1050 is impressive compared to other models.

ANN single hidden layer

In [281]:
```
tf.reset_default_graph()
def weight_bais():
    global weights, biases
    weights = {
        'w1': tf.Variable(tf.random_normal([input_dim, hidden_1])),
        'out': tf.Variable(tf.random_normal([hidden_1, output_dim]))
    }
    biases = {
        'b1': tf.Variable(tf.random_normal([hidden_1])),
        'out': tf.Variable(tf.random_normal([output_dim]))
    }
```

In [282]:
```
def ann_model(X_val):
  # Hidden layers
    layer_1 = tf.add(tf.matmul(X_val, weights['w1']), biases['b1'])
    layer_1 = tf.nn.relu(layer_1)

    # Output layer
    layer_out = tf.add(tf.matmul(layer_1, weights['out']), biases['out'])

    return layer_out
```

In [283]:
```
def miscellaneous_initialization():
    global model, cost , regularizer_1 , regularizer_2 ,regularizer_3, regularizer_4, optimizer , init , saver
    # Model Construct
    model = ann_model(X_tf)

    # Mean Squared Error cost function
    cost = tf.reduce_mean(tf.square(y_tf - model))

    # cost = tf.square(y_tf - model)
    regularizer_1 = tf.nn.l2_loss(weights['w1'])

    # cost = tf.reduce_mean(cost + beta1*regularizer_1 + beta2*regularizer_2 + beta3*regularizer_3)
    cost = cost + beta1*regularizer_1

    # Adam optimizer will update weights and biases after each step
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)


    # Initialize variables
    init = tf.global_variables_initializer()

    # Add ops to save and restore all the variables.
    saver = tf.train.Saver()
```

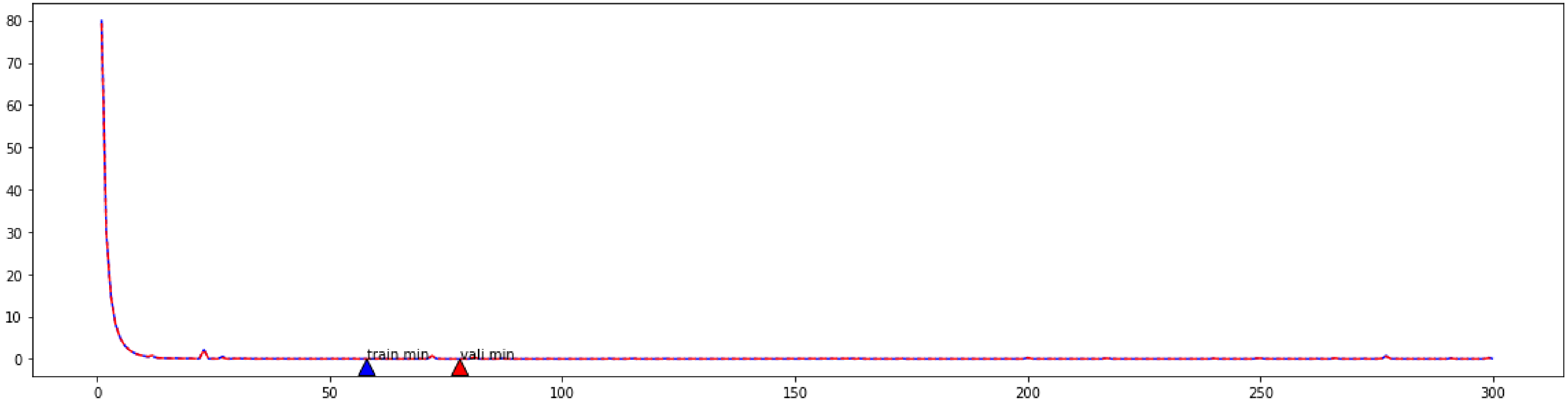
ANN 4

- learning rate = .1

layer name	Neuron	value of beta for l2 regularization
1st hidden layer	16 Neuron	.1

In [284]:
```
tf.reset_default_graph()
learning_rate = 0.1
num_steps = 15000
#for regularize weight matrix
beta1 = 0.1
beta2 = None
beta3 = None
beta4 = None
minimum_validation_cost = 0.01901000
hidden_1 = 16
hidden_2 = None
hidden_3 = None
hidden_4 = None

#tf graph input
X_tf = tf.placeholder("float" )
y_tf = tf.placeholder("float" )



weight_bais()
miscellaneous_initialization()
train_LC = []
val_LC = []
training_block(X_train,y_train, X_test,y_test)
prediction = Prediction_block(X_test)

test_rmse_score, test_r2_score = accuracy(y_test,prediction)

print('ann root mean absolute error: ', test_rmse_score)
print('accuracy score: ', test_r2_score  )
learning_curve()


pred_str = 'ANN_lr'+str(learning_rate)+'_beta'+str(beta1)+'-'+str(beta2)+'-'+str(beta3)+'-'+str(beta4)+'_hidden'+str(hidden_1)+'-
'+str(hidden_2)+'-'+str(hidden_3)+'-'+str(hidden_4)
prediction_dict[pred_str] = prediction

if submit:
    submit_prediction = Prediction_block(test_processed)
    submit_prediction_dict[pred_str] = submit_prediction
# Data Save
if save_score:
    log_df = pd.read_csv("diffrent_training_results.csv")
    log_df = log_df.append({'learning_rate' : learning_rate, 'num_steps' : num_steps, 'beta1' : beta1, 'beta2' : beta2, 'beta3' :
beta3, 'beta4' : beta4, 'hidden_1' : hidden_1 , 'hidden_2' : hidden_2, 'hidden_3' : hidden_3, 'hidden_4' : hidden_4, 'input_dim'
: input_dim , 'test_rmse_score' : test_rmse_score , 'test_r2_score' : test_r2_score}, ignore_index=True)
    log_df.to_csv("diffrent_training_results.csv", encoding='utf-8',index=False)
```

```
epoch no :  500   training cost:  0.70696795   validation cost:  0.70672375     minimum_validation_cost 0.01901
epoch no :  1000   training cost:  0.074884504   validation cost:  0.09517263     minimum_validation_cost 0.01901
epoch no :  1500   training cost:  0.030514441   validation cost:  0.037706994     minimum_validation_cost 0.01901
epoch no :  2000   training cost:  0.041105207   validation cost:  0.06775436     minimum_validation_cost 0.0175587
epoch no :  2500   training cost:  0.040804587   validation cost:  0.048201256     minimum_validation_cost 0.015757574
epoch no :  3000   training cost:  0.023604002   validation cost:  0.020474989     minimum_validation_cost 0.01474772
epoch no :  3500   training cost:  0.021396361   validation cost:  0.02962465     minimum_validation_cost 0.013838205
epoch no :  4000   training cost:  0.011788214   validation cost:  0.013150874     minimum_validation_cost 0.013117214
epoch no :  4500   training cost:  0.011998331   validation cost:  0.013331305     minimum_validation_cost 0.013110494
epoch no :  5000   training cost:  0.01817273   validation cost:  0.016207738     minimum_validation_cost 0.013110494
epoch no :  5500   training cost:  0.107278176   validation cost:  0.05767396     minimum_validation_cost 0.013110494
epoch no :  6000   training cost:  0.019238576   validation cost:  0.016842552     minimum_validation_cost 0.013110494
epoch no :  6500   training cost:  0.013488971   validation cost:  0.015036552     minimum_validation_cost 0.013110494
epoch no :  7000   training cost:  0.014104144   validation cost:  0.014767384     minimum_validation_cost 0.013110494
epoch no :  7500   training cost:  0.0142570175   validation cost:  0.015163885     minimum_validation_cost 0.013110494
epoch no :  8000   training cost:  0.014391301   validation cost:  0.015554772     minimum_validation_cost 0.013110494
epoch no :  8500   training cost:  0.03745286   validation cost:  0.047344368     minimum_validation_cost 0.013110494
epoch no :  9000   training cost:  0.015391441   validation cost:  0.017085707     minimum_validation_cost 0.013110494
epoch no :  9500   training cost:  0.05376236   validation cost:  0.02105463     minimum_validation_cost 0.013110494
epoch no :  10000   training cost:  0.2804591   validation cost:  0.29511937     minimum_validation_cost 0.013110494
epoch no :  10500   training cost:  0.015953306   validation cost:  0.016810209     minimum_validation_cost 0.013110494
epoch no :  11000   training cost:  0.01592241   validation cost:  0.016748453     minimum_validation_cost 0.013110494
epoch no :  11500   training cost:  0.01610453   validation cost:  0.01695231     minimum_validation_cost 0.013110494
epoch no :  12000   training cost:  0.07394206   validation cost:  0.12320263     minimum_validation_cost 0.013110494
epoch no :  12500   training cost:  0.15734634   validation cost:  0.15152527     minimum_validation_cost 0.013110494
epoch no :  13000   training cost:  0.01652914   validation cost:  0.017331216     minimum_validation_cost 0.013110494
epoch no :  13500   training cost:  0.016681496   validation cost:  0.017460056     minimum_validation_cost 0.013110494
epoch no :  14000   training cost:  0.020754531   validation cost:  0.021394152     minimum_validation_cost 0.013110494
epoch no :  14500   training cost:  0.016063426   validation cost:  0.016903006     minimum_validation_cost 0.013110494
epoch no :  15000   training cost:  0.019355612   validation cost:  0.020434875     minimum_validation_cost 0.013110494
Model saved in path: model/model.ckpt
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
ann root mean absolute error:  0.10595639845030991
accuracy score:  0.9337554952955662
```



ANN 5

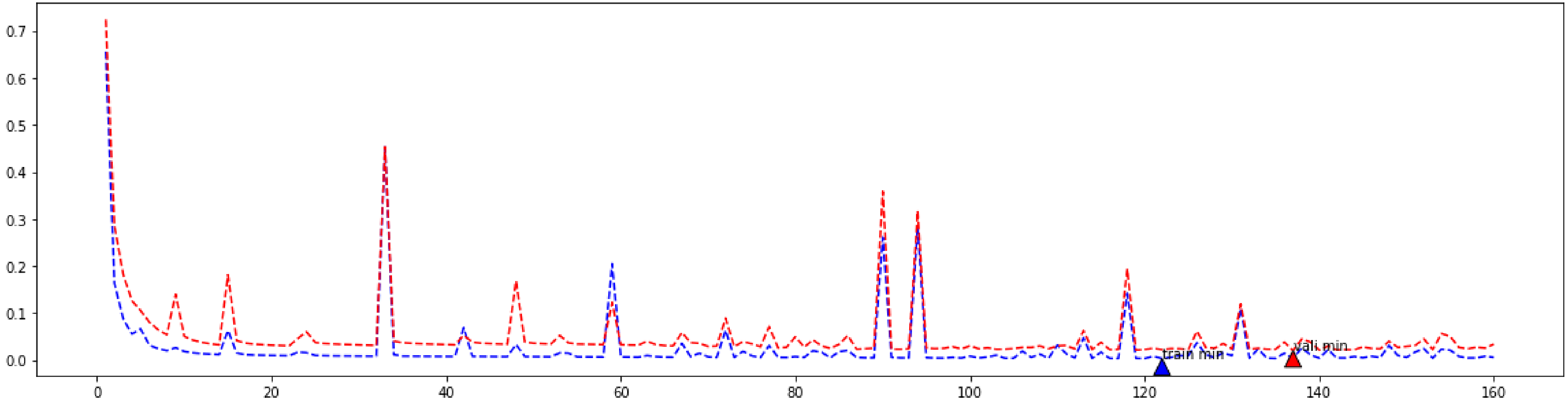
- learning rate = .1

layer name	Neuron	value of beta for l2 regularization
1st hidden layer	4 Neuron	.1

In [285]:
```
tf.reset_default_graph()
learning_rate = 0.1
num_steps = 8000
#for regularize weight matrix
beta1 = 0
beta2 = None
beta3 = None
beta4 = None

hidden_1 = 4
hidden_2 = None
hidden_3 = None
hidden_4 = None
minimum_validation_cost = 0.1701000
#tf graph input
X_tf = tf.placeholder("float" )
y_tf = tf.placeholder("float" )


weight_bais()
miscellaneous_initialization()
train_LC = []
val_LC = []
```

In [286]:
```
training_block(X_train,y_train, X_test,y_test)
prediction = Prediction_block(X_test)

test_rmse_score, test_r2_score = accuracy(y_test,prediction)

print('ann root mean absolute error: ', test_rmse_score)
print('accuracy score: ', test_r2_score  )
learning_curve()

pred_str = 'ANN_lr'+str(learning_rate)+'_beta'+str(beta1)+'-'+str(beta2)+'-'+str(beta3)+'-'+str(beta4)+'_hidden'+str(hidden_1)+'-
'+str(hidden_2)+'-'+str(hidden_3)+'-'+str(hidden_4)
prediction_dict[pred_str] = prediction

if submit:
    submit_prediction = Prediction_block(test_processed)
    submit_prediction_dict[pred_str] = submit_prediction
# Data Save
if save_score:
    log_df = pd.read_csv("diffrent_training_results.csv")
    log_df = log_df.append({'learning_rate' : learning_rate, 'num_steps' : num_steps, 'beta1' : beta1, 'beta2' : beta2, 'beta3' :
beta3, 'beta4' : beta4, 'hidden_1' : hidden_1 , 'hidden_2' : hidden_2, 'hidden_3' : hidden_3, 'hidden_4' : hidden_4, 'input_dim'
: input_dim , 'test_rmse_score' : test_rmse_score , 'test_r2_score' : test_r2_score}, ignore_index=True)
    log_df.to_csv("diffrent_training_results.csv", encoding='utf-8',index=False)
```

```
epoch no :  500   training cost:  0.018612064   validation cost:  0.050663006     minimum_validation_cost 0.05027496
epoch no :  1000   training cost:  0.010205073   validation cost:  0.03200774     minimum_validation_cost 0.032025207
epoch no :  1500   training cost:  0.009000993   validation cost:  0.03263801     minimum_validation_cost 0.030169504
epoch no :  2000   training cost:  0.008385307   validation cost:  0.033433408     minimum_validation_cost 0.030169504
epoch no :  2500   training cost:  0.007881007   validation cost:  0.03631349     minimum_validation_cost 0.030169504
epoch no :  3000   training cost:  0.0072417823   validation cost:  0.03308853     minimum_validation_cost 0.030169504
epoch no :  3500   training cost:  0.0074811154   validation cost:  0.029426454     minimum_validation_cost 0.029408183
epoch no :  4000   training cost:  0.008039351   validation cost:  0.050030284     minimum_validation_cost 0.026420904
epoch no :  4500   training cost:  0.26158154   validation cost:  0.35876465     minimum_validation_cost 0.02241918
epoch no :  5000   training cost:  0.00908185   validation cost:  0.030987838     minimum_validation_cost 0.021854304
epoch no :  5500   training cost:  0.032166183   validation cost:  0.030488687     minimum_validation_cost 0.02147018
epoch no :  6000   training cost:  0.003865236   validation cost:  0.022981284     minimum_validation_cost 0.0201902
epoch no :  6500   training cost:  0.0096211   validation cost:  0.023849959     minimum_validation_cost 0.0201902
epoch no :  7000   training cost:  0.0046753194   validation cost:  0.021406047     minimum_validation_cost 0.019293755
epoch no :  7500   training cost:  0.0064295623   validation cost:  0.029499253     minimum_validation_cost 0.019293755
epoch no :  8000   training cost:  0.006344047   validation cost:  0.03422031     minimum_validation_cost 0.019293755
Model saved in path: model/model.ckpt
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
ann root mean absolute error:  0.13890197207722216
accuracy score:  0.8861554341828791
```



ANN 6

- learning rate = .1

layer name	Neuron	value of beta for l2 regularization
1st hidden layer	2 Neuron	.1

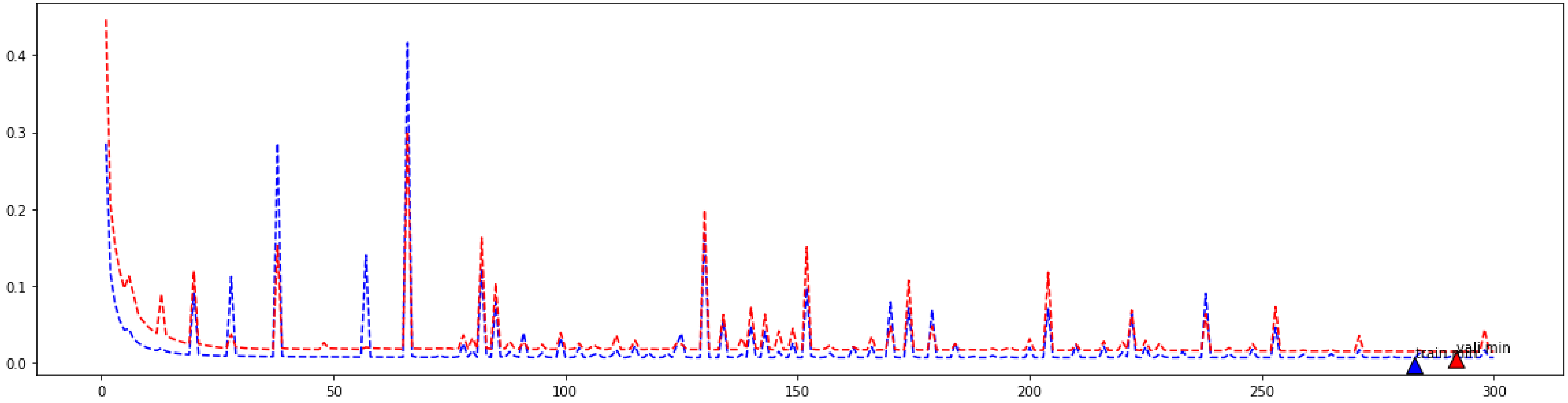
In [98]:

```
tf.reset_default_graph()
learning_rate = 0.1
num_steps = 15000
#for regularize weight matrix
beta1 = 0
beta2 = None
beta3 = None
beta4 = None

hidden_1 = 2
hidden_2 = None
hidden_3 = None
hidden_4 = None
minimum_validation_cost = 0.01901000
#tf graph input
X_tf = tf.placeholder("float" )
y_tf = tf.placeholder("float" )


weight_bais()
miscellaneous_initialization()
train_LC = []
val_LC = []
```

In [99]:

```
training_block(X_train,y_train, X_test,y_test)
prediction = Prediction_block(X_test)

test_rmse_score, test_r2_score = accuracy(y_test,prediction)

print('ann root mean absolute error: ', test_rmse_score)
print('accuracy score: ', test_r2_score  )
learning_curve()

pred_str = 'ANN_lr'+str(learning_rate)+'_beta'+str(beta1)+'-'+str(beta2)+'-'+str(beta3)+'-'+str(beta4)+'_hidden'+str(hidden_1)+'-
'+str(hidden_2)+'-'+str(hidden_3)+'-'+str(hidden_4)
prediction_dict[pred_str] = prediction

if submit:
    submit_prediction = Prediction_block(test_processed)
    submit_prediction_dict[pred_str] = submit_prediction
# Data Save
if save_score:
    log_df = pd.read_csv("diffrent_training_results.csv")
    log_df = log_df.append({'learning_rate' : learning_rate, 'num_steps' : num_steps, 'beta1' : beta1, 'beta2' : beta2, 'beta3' :
beta3, 'beta4' : beta4, 'hidden_1' : hidden_1 , 'hidden_2' : hidden_2, 'hidden_3' : hidden_3, 'hidden_4' : hidden_4, 'input_dim'
: input_dim , 'test_rmse_score' : test_rmse_score , 'test_r2_score' : test_r2_score}, ignore_index=True)
    log_df.to_csv("diffrent_training_results.csv", encoding='utf-8',index=False)
```

```
epoch no :  500   training cost:  0.019526327   validation cost:  0.04785805    minimum_validation_cost 0.01901
epoch no :  1000   training cost:  0.090430334   validation cost:  0.11915722    minimum_validation_cost 0.01901
epoch no :  1500   training cost:  0.009021159   validation cost:  0.019854361    minimum_validation_cost 0.01901
epoch no :  2000   training cost:  0.008328172   validation cost:  0.018638173    minimum_validation_cost 0.017471917
epoch no :  2500   training cost:  0.007971583   validation cost:  0.018678952    minimum_validation_cost 0.017459333
epoch no :  3000   training cost:  0.0077294353   validation cost:  0.018802863    minimum_validation_cost 0.017459333
epoch no :  3500   training cost:  0.0075098113   validation cost:  0.018267034    minimum_validation_cost 0.017459333
epoch no :  4000   training cost:  0.015966242   validation cost:  0.032293662    minimum_validation_cost 0.017459333
epoch no :  4500   training cost:  0.0074046743   validation cost:  0.017824942    minimum_validation_cost 0.017459333
epoch no :  5000   training cost:  0.00737159   validation cost:  0.017943176    minimum_validation_cost 0.017459333
epoch no :  5500   training cost:  0.009448293   validation cost:  0.018865194    minimum_validation_cost 0.017379621
epoch no :  6000   training cost:  0.007176871   validation cost:  0.017734537    minimum_validation_cost 0.017366491
epoch no :  6500   training cost:  0.17015442   validation cost:  0.19888674    minimum_validation_cost 0.017291883
epoch no :  7000   training cost:  0.048054833   validation cost:  0.07152697    minimum_validation_cost 0.017291883
epoch no :  7500   training cost:  0.007178404   validation cost:  0.017213237    minimum_validation_cost 0.017266192
epoch no :  8000   training cost:  0.0071287826   validation cost:  0.017250806    minimum_validation_cost 0.017099544
epoch no :  8500   training cost:  0.078824885   validation cost:  0.047651436    minimum_validation_cost 0.016776746
epoch no :  9000   training cost:  0.007192865   validation cost:  0.017091228    minimum_validation_cost 0.016776746
epoch no :  9500   training cost:  0.0071183126   validation cost:  0.016838994    minimum_validation_cost 0.016650783
epoch no :  10000   training cost:  0.020033514   validation cost:  0.03116488    minimum_validation_cost 0.016502209
epoch no :  10500   training cost:  0.023244442   validation cost:  0.024413731    minimum_validation_cost 0.01633817
epoch no :  11000   training cost:  0.013847738   validation cost:  0.027082304    minimum_validation_cost 0.016238058
epoch no :  11500   training cost:  0.007122549   validation cost:  0.016189996    minimum_validation_cost 0.015966171
epoch no :  12000   training cost:  0.007111148   validation cost:  0.0159556    minimum_validation_cost 0.015759723
epoch no :  12500   training cost:  0.007110614   validation cost:  0.01570639    minimum_validation_cost 0.01548126
epoch no :  13000   training cost:  0.0071250037   validation cost:  0.0154177975    minimum_validation_cost 0.0152186025
epoch no :  13500   training cost:  0.0071092476   validation cost:  0.015325032    minimum_validation_cost 0.015116011
epoch no :  14000   training cost:  0.007109693   validation cost:  0.015189837    minimum_validation_cost 0.014980882
epoch no :  14500   training cost:  0.007109804   validation cost:  0.015131356    minimum_validation_cost 0.014810974
epoch no :  15000   training cost:  0.007110104   validation cost:  0.014890727    minimum_validation_cost 0.014607174
Model saved in path: model/model.ckpt
INFO:tensorflow:Restoring parameters from model/model.ckpt
Model restored.
ann root mean absolute error:  0.12086015230808778
accuracy score:  0.9138090324301579
```



Description on Learning curve and Accuracy:

We can observe where overfitting occurs. Overfitting actually occurs if the training loss goes under the validation loss even though the validation is still dropping. It is the sign that network is learning the patterns in the train set that are not applicable in the validation done. In a short note we can say::

Overfitting : training loss << validation loss

Underfitting : training loss >> validation loss

Just right : training loss ~ validation loss

According to this theory, for ANN 4 our both learning curve (validation cost and training cost) is exactly top of one another so in our case validation cost and training cost is almost same so we can say that our model is doing just the right thing. Again In validation score .1059 is impressive compared to other models.

But for ANN 5 and 6 training loss << validation loss so we can say that this two model overfit data due to lower amount of neuron but ANN 4 have just the right amount of neuron thats why with similar parameter this overfit occered.

Hyperparameeter tuning

Few of my hyperparameeter tuning is shown in the following block. In this data if a hidden layer value is 0 then it means that the hidden layer is turned off. For example if hidden_3 = 0 then that means hidden layer 3 is removed from the model and the model have only 2 hidden layer. And all the score is done on a validation set which is not seen by the model while training. For most of the case it was a 80-20 split. In the following results I didint kept any cross validation results but I have used diffrent seed while splitting data due to diffrent seed sometimes good hyperparameeter also provided so so accuracy.

In [255]:
```
log_df = pd.read_csv("diffrent_training_results.csv")
# print(log_df.to_string())
pd.set_option('display.max_rows', None)
log_df
```

Out[255]:

	learning_rate	num_steps	beta1	beta2	beta3	hidden_1	hidden_2	hidden_3	input_dim	test_rmse_score	test_r2_score	hidden_4	beta4
0	0.050	7500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.128456	8.949361e-01	NaN	NaN
1	0.040	2500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.250470	6.005558e-01	NaN	NaN
2	0.050	7500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.152580	8.517686e-01	NaN	NaN
3	0.050	7500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.143409	8.690530e-01	NaN	NaN
4	0.050	7500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.127356	8.967284e-01	NaN	NaN
5	0.050	7900.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.126758	8.976948e-01	NaN	NaN
6	0.050	1500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.162495	8.318785e-01	NaN	NaN
7	0.050	1500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.177628	7.991050e-01	NaN	NaN
8	0.050	7500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.139909	8.753655e-01	NaN	NaN
9	0.050	7500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.143775	8.683835e-01	NaN	NaN
10	0.100	7500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.138477	8.779036e-01	NaN	NaN
11	0.100	7500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.138477	8.779036e-01	NaN	NaN
12	0.010	7500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.154219	8.485668e-01	NaN	NaN
13	0.010	7500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.154219	8.485668e-01	NaN	NaN
14	0.010	7500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.661086	-1.782662e+00	NaN	NaN
15	0.100	3500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.131423	8.900259e-01	NaN	NaN
16	0.100	2800.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.389771	-5.111744e-03	NaN	NaN
17	0.100	2500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.390269	-7.679426e-03	NaN	NaN
18	0.100	8500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.102641	9.302995e-01	NaN	NaN
19	0.050	8500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.135519	8.830642e-01	NaN	NaN
20	0.100	2500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.304550	4.094407e-01	NaN	NaN
21	0.100	9500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.124793	9.008422e-01	NaN	NaN
22	0.001	9500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	14.246912	-1.291369e+03	NaN	NaN
23	0.005	9500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.710445	-2.213707e+00	NaN	NaN
24	0.100	7500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.129652	8.929701e-01	NaN	NaN
25	0.100	12500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.142223	8.712101e-01	NaN	NaN
26	0.100	11500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.126000	8.989146e-01	NaN	NaN
27	0.050	11500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.124008	9.020864e-01	NaN	NaN
28	0.050	11000.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.147886	8.607480e-01	NaN	NaN
29	0.050	11000.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.105610	9.262080e-01	NaN	NaN
30	0.050	11000.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.131099	8.905686e-01	NaN	NaN
31	0.010	11000.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	2.365656	-3.463267e+01	NaN	NaN

	learning_rate	num_steps	beta1	beta2	beta3	hidden_1	hidden_2	hidden_3	input_dim	test_rmse_score	test_r2_score	hidden_4	beta4
32	0.100	11000.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.135869	8.824604e-01	NaN	NaN
33	0.100	11000.0	0.005	0.0050	0.000000	16.0	8.0	4.0	403.0	0.132422	8.883490e-01	NaN	NaN
34	0.050	13000.0	0.005	0.0050	0.000000	16.0	8.0	4.0	403.0	0.126124	8.987160e-01	NaN	NaN
35	0.100	23000.0	0.005	0.0050	0.000000	16.0	8.0	4.0	403.0	0.123945	9.021856e-01	NaN	NaN
36	0.100	23000.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.321211	3.430598e-01	NaN	NaN
37	0.050	23000.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.126705	8.977802e-01	NaN	NaN
38	0.050	23000.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.126705	8.977802e-01	NaN	NaN
39	0.050	23000.0	0.000	0.0000	0.000000	16.0	8.0	4.0	403.0	0.173484	8.083707e-01	NaN	NaN
40	0.050	23000.0	0.000	0.0000	0.000000	16.0	8.0	4.0	403.0	0.396362	-2.986747e-04	NaN	NaN
41	0.100	23000.0	0.000	0.0000	0.000000	16.0	8.0	4.0	403.0	0.396368	-3.297183e-04	NaN	NaN
42	0.100	17000.0	0.000	0.0000	0.000000	16.0	8.0	4.0	403.0	0.396362	-2.981759e-04	NaN	NaN
43	0.001	3000.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	2.654960	-4.388086e+01	NaN	NaN
44	0.001	3000.0	0.050	0.0000	0.000000	16.0	8.0	4.0	403.0	7.177147	-3.269811e+02	NaN	NaN
45	0.100	3000.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.124692	9.010024e-01	NaN	NaN
46	0.100	3000.0	0.010	0.0000	0.000000	16.0	8.0	4.0	403.0	0.126720	8.977569e-01	NaN	NaN
47	0.100	13000.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.396362	-2.986747e-04	NaN	NaN
48	0.100	2500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.136049	8.821482e-01	NaN	NaN
49	0.100	3000.0	0.100	0.1000	0.000000	16.0	8.0	4.0	403.0	0.127360	8.967204e-01	NaN	NaN
50	0.100	4000.0	0.100	0.1000	0.000000	16.0	8.0	4.0	403.0	0.260455	5.680732e-01	NaN	NaN
51	0.100	3500.0	0.100	0.0100	0.000000	16.0	8.0	4.0	403.0	0.146195	8.639153e-01	NaN	NaN
52	0.100	3500.0	0.100	0.0010	0.000000	16.0	8.0	4.0	403.0	0.396370	-3.376305e-04	NaN	NaN
53	0.100	3500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.113379	9.149525e-01	NaN	NaN
54	0.100	3500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.117934	9.079823e-01	NaN	NaN
55	0.100	8500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.111485	9.177711e-01	NaN	NaN
56	0.100	7600.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.106919	9.243687e-01	NaN	NaN
57	0.050	7600.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.119165	9.060501e-01	NaN	NaN
58	0.100	7600.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.376772	6.081402e-02	NaN	NaN
59	0.100	3500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.107788	9.231341e-01	NaN	NaN
60	0.050	8500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.122025	9.014877e-01	NaN	NaN
61	0.100	7600.0	0.005	0.0050	0.000000	16.0	8.0	4.0	403.0	0.389771	-5.109640e-03	NaN	NaN
62	0.100	7600.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.109705	9.203748e-01	NaN	NaN
63	0.100	7600.0	0.100	0.0050	0.005000	200.0	100.0	30.0	403.0	0.389676	-4.622793e-03	NaN	NaN

	learning_rate	num_steps	beta1	beta2	beta3	hidden_1	hidden_2	hidden_3	input_dim	test_rmse_score	test_r2_score	hidden_4	beta4
64	0.100	9600.0	0.000	0.0000	0.000000	200.0	100.0	30.0	403.0	0.389825	-5.391521e-03	NaN	NaN
65	0.100	3600.0	0.100	0.0000	0.000000	200.0	100.0	30.0	403.0	1.764792	-1.960547e+01	NaN	NaN
66	0.100	7600.0	0.100	0.0000	0.000000	200.0	100.0	30.0	403.0	0.389774	-5.127892e-03	NaN	NaN
67	0.100	17600.0	0.100	0.0000	0.000000	200.0	100.0	30.0	403.0	0.389772	-5.119113e-03	NaN	NaN
68	0.100	15600.0	0.100	0.0000	0.000000	200.0	100.0	30.0	403.0	0.149463	8.522031e-01	NaN	NaN
69	0.100	15600.0	0.100	0.0000	0.000000	32.0	16.0	8.0	403.0	0.389750	-5.006016e-03	NaN	NaN
70	0.100	3600.0	0.100	0.0000	0.000000	1.0	0.0	0.0	403.0	0.104633	9.275680e-01	NaN	NaN
71	0.100	7500.0	0.100	0.0000	0.000000	1.0	0.0	0.0	403.0	0.105433	9.264551e-01	NaN	NaN
72	0.100	7500.0	0.010	0.0000	0.000000	1.0	0.0	0.0	403.0	0.104463	9.278026e-01	NaN	NaN
73	0.100	6000.0	0.010	0.0000	0.000000	1.0	0.0	0.0	403.0	0.158149	8.345259e-01	NaN	NaN
74	0.100	86000.0	0.010	0.0000	0.000000	1.0	0.0	0.0	403.0	0.389882	-5.682449e-03	NaN	NaN
75	0.100	8600.0	0.010	0.0000	0.000000	1.0	0.0	0.0	403.0	0.389771	-5.109640e-03	NaN	NaN
76	0.100	8600.0	0.100	0.0000	0.000000	1.0	0.0	0.0	403.0	0.111076	9.183720e-01	NaN	NaN
77	0.100	3600.0	0.100	0.0000	0.000000	4.0	0.0	0.0	403.0	0.105270	9.266836e-01	NaN	NaN
78	0.100	3600.0	0.100	0.0000	0.000000	1.0	0.0	0.0	403.0	0.104898	9.271996e-01	NaN	NaN
79	0.100	3600.0	0.100	0.0000	0.000000	32.0	0.0	0.0	403.0	0.198056	7.404795e-01	NaN	NaN
80	0.100	3600.0	0.100	0.0000	0.000000	16.0	0.0	0.0	403.0	0.607723	-1.443464e+00	NaN	NaN
81	0.100	3600.0	0.100	0.0000	0.000000	16.0	0.0	0.0	403.0	0.106043	9.256030e-01	NaN	NaN
82	0.100	7600.0	0.100	0.0000	0.000000	1.0	0.0	0.0	403.0	0.105914	9.257826e-01	NaN	NaN
83	0.100	7600.0	0.100	0.0000	0.000000	16.0	0.0	0.0	403.0	0.107050	9.241823e-01	NaN	NaN
84	0.100	3900.0	0.100	0.0000	0.000000	16.0	0.0	0.0	403.0	0.107679	9.232893e-01	NaN	NaN
85	0.100	2000.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.157928	8.411961e-01	NaN	NaN
86	0.100	7600.0	0.100	0.0050	0.005000	16.0	8.0	4.0	403.0	0.138570	8.777409e-01	NaN	NaN
87	0.100	8600.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.133445	8.866164e-01	NaN	NaN
88	0.100	8600.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.129291	8.935660e-01	NaN	NaN
89	0.100	3600.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.119097	9.096871e-01	NaN	NaN
90	0.100	3500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.512748	-6.739936e-01	NaN	NaN
91	0.100	9600.0	0.100	0.0050	0.005000	16.0	8.0	4.0	403.0	0.124896	9.006785e-01	NaN	NaN
92	0.100	9600.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.205646	7.307324e-01	NaN	NaN
93	0.100	19600.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.132143	8.888180e-01	NaN	NaN
94	0.100	19600.0	0.100	0.0100	0.001000	16.0	8.0	4.0	403.0	0.235628	6.464929e-01	NaN	NaN
95	0.100	19600.0	0.100	0.0005	0.000005	16.0	8.0	4.0	403.0	0.128857	8.942789e-01	NaN	NaN

	learning_rate	num_steps	beta1	beta2	beta3	hidden_1	hidden_2	hidden_3	input_dim	test_rmse_score	test_r2_score	hidden_4	beta4
96	0.100	19600.0	0.000	0.0000	0.000000	16.0	8.0	4.0	403.0	0.396391	-4.451594e-04	NaN	NaN
97	0.100	29600.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.148700	8.592123e-01	NaN	NaN
98	0.100	4000.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.172257	8.110704e-01	NaN	NaN
99	0.100	1750.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.396362	-2.980097e-04	NaN	NaN
100	0.100	3600.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.127960	8.957456e-01	NaN	NaN
101	0.100	4000.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.122812	9.039655e-01	NaN	NaN
102	0.100	7600.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.127266	8.968736e-01	NaN	NaN
103	0.100	5500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.127831	8.959562e-01	NaN	NaN
104	0.100	7600.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	399.635052	-1.016885e+06	NaN	NaN
105	0.100	17600.0	0.100	0.0005	0.000005	16.0	8.0	4.0	403.0	0.122934	9.037754e-01	NaN	NaN
106	0.100	17100.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.123996	9.021051e-01	NaN	NaN
107	0.100	19500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.122128	9.050330e-01	NaN	NaN
108	0.100	29500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.121803	9.055369e-01	NaN	NaN
109	0.100	49500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.120712	9.072223e-01	NaN	NaN
110	0.100	9500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.104178	9.281964e-01	NaN	NaN
111	0.100	19500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.089601	9.468848e-01	NaN	NaN
112	0.100	19500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.118142	9.076568e-01	NaN	NaN
113	0.100	19500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.118142	9.076568e-01	NaN	NaN
114	0.100	9500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.090853	9.453897e-01	NaN	NaN
115	0.100	9500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.090915	9.453158e-01	NaN	NaN
116	0.100	19500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.091638	9.444420e-01	NaN	NaN
117	0.050	39500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.076999	9.607748e-01	NaN	NaN
118	0.100	49500.0	0.005	0.0050	0.005000	16.0	8.0	4.0	403.0	0.073579	9.641818e-01	NaN	NaN
119	0.100	49500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.087605	9.492243e-01	NaN	NaN
120	0.100	9500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.159089	8.325543e-01	NaN	NaN
121	0.100	29500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.093281	9.424324e-01	NaN	NaN
122	0.100	29500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.127969	8.689350e-01	NaN	NaN
123	0.100	9500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.117637	8.892446e-01	NaN	NaN
124	0.100	9500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.104068	9.283477e-01	NaN	NaN
125	0.100	9500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.103677	9.288849e-01	NaN	NaN
126	0.100	9500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.103494	9.291363e-01	NaN	NaN
127	0.100	9500.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.103775	9.364546e-01	NaN	NaN

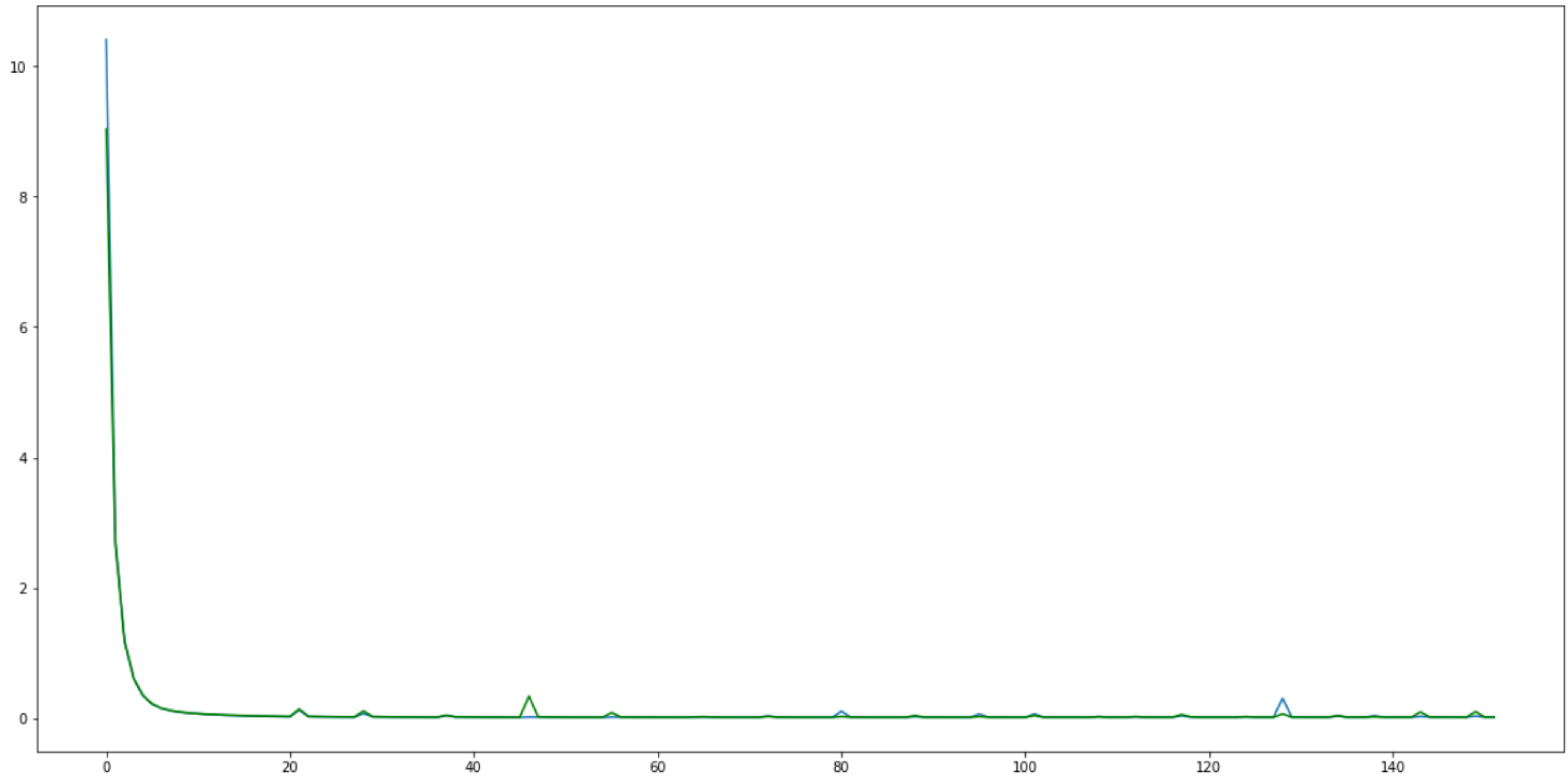
	learning_rate	num_steps	beta1	beta2	beta3	hidden_1	hidden_2	hidden_3	input_dim	test_rmse_score	test_r2_score	hidden_4	beta4
128	0.100	26000.0	0.100	0.0000	0.000000	200.0	100.0	50.0	403.0	0.112221	9.256909e-01	NaN	NaN
129	0.100	26000.0	0.100	0.0000	0.000000	200.0	100.0	50.0	403.0	0.111169	9.270777e-01	12.0	NaN
130	0.100	26000.0	0.100	0.0000	0.000000	200.0	100.0	50.0	403.0	0.111169	9.270777e-01	12.0	NaN
131	0.100	46000.0	0.100	0.0000	0.000000	200.0	100.0	50.0	403.0	0.411770	-4.727709e-04	12.0	NaN
132	0.100	26000.0	0.000	0.0000	0.000000	200.0	96.0	32.0	403.0	0.411734	-2.970530e-04	4.0	NaN
133	0.100	6000.0	0.000	0.0000	0.000000	200.0	96.0	32.0	403.0	0.412326	-3.176449e-03	4.0	0.0
134	0.100	26000.0	0.100	0.0000	0.000000	200.0	96.0	32.0	403.0	0.411762	-4.307332e-04	4.0	0.0
135	0.100	26000.0	0.100	0.0000	0.000000	32.0	16.0	8.0	403.0	0.411745	-3.482792e-04	4.0	0.0
136	0.100	46000.0	0.100	0.1000	0.000000	200.0	100.0	50.0	403.0	0.411750	-3.748811e-04	25.0	0.0
137	0.100	46000.0	0.100	0.1000	0.000000	200.0	100.0	50.0	403.0	0.411750	-3.748811e-04	25.0	0.0
138	0.100	26000.0	0.100	0.0000	0.000000	200.0	100.0	50.0	403.0	0.411739	-3.223233e-04	12.0	0.0
139	0.050	26000.0	0.100	0.0000	0.000000	200.0	100.0	50.0	403.0	0.412324	-3.166794e-03	12.0	0.0
140	0.100	15000.0	0.100	NaN	NaN	16.0	NaN	NaN	403.0	0.115072	9.218665e-01	NaN	NaN
141	0.100	15000.0	0.000	NaN	NaN	1.0	NaN	NaN	403.0	0.116185	9.203481e-01	NaN	NaN
142	0.100	15000.0	0.000	NaN	NaN	2.0	NaN	NaN	403.0	0.150994	8.654721e-01	NaN	NaN
143	0.100	15000.0	NaN	NaN	NaN	NaN	NaN	NaN	403.0	0.162648	8.439027e-01	NaN	NaN
144	0.100	35000.0	NaN	NaN	NaN	NaN	NaN	NaN	403.0	0.147050	8.724072e-01	NaN	NaN
145	0.100	8000.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.184813	7.984608e-01	NaN	NaN
146	0.100	20000.0	0.100	0.0500	0.000000	76.0	48.0	32.0	403.0	0.111441	9.267196e-01	16.0	0.0
147	0.100	25000.0	0.100	0.0500	0.000000	128.0	64.0	32.0	403.0	0.412339	-3.237055e-03	16.0	0.0
148	0.100	8000.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.111108	9.271576e-01	NaN	NaN
149	0.100	25000.0	0.100	0.0500	0.000000	128.0	64.0	32.0	403.0	0.150787	8.658403e-01	16.0	0.0
150	0.050	35000.0	0.100	0.0000	0.000000	128.0	64.0	32.0	403.0	0.412324	-3.166794e-03	16.0	0.0
151	0.100	25000.0	0.100	0.0000	0.000000	256.0	128.0	32.0	403.0	0.412324	-3.166794e-03	8.0	0.0
152	0.050	25000.0	0.100	0.0000	0.000000	256.0	128.0	32.0	403.0	0.412324	-3.167316e-03	8.0	0.0
153	0.100	15000.0	0.100	0.0000	0.000000	4.0	16.0	16.0	403.0	0.412324	-3.167055e-03	4.0	0.0
154	0.100	35000.0	0.100	0.0000	0.000000	256.0	128.0	64.0	403.0	0.412325	-3.170185e-03	32.0	0.0
155	0.100	25000.0	0.100	0.1000	0.000000	256.0	128.0	32.0	403.0	0.123334	9.102441e-01	8.0	0.0
156	0.050	25000.0	0.100	0.0000	0.000000	256.0	128.0	64.0	403.0	0.412324	-3.167055e-03	8.0	0.0
157	0.050	25000.0	0.100	0.0000	0.000000	128.0	64.0	16.0	403.0	0.412319	-3.141554e-03	4.0	0.0
158	0.100	35000.0	0.100	0.0000	0.000000	16.0	32.0	48.0	403.0	0.416205	-2.213767e-02	76.0	0.0
159	0.100	15000.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.412324	-3.166794e-03	2.0	0.0

	learning_rate	num_steps	beta1	beta2	beta3	hidden_1	hidden_2	hidden_3	input_dim	test_rmse_score	test_r2_score	hidden_4	beta4
160	0.050	25000.0	0.100	0.0500	0.000000	128.0	64.0	16.0	403.0	0.412324	-3.167837e-03	4.0	0.0
161	0.050	25000.0	0.100	0.0000	0.000000	190.0	90.0	30.0	403.0	0.412324	-3.165491e-03	3.0	0.0
162	0.100	8000.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.412324	-3.166794e-03	2.0	0.0
163	0.100	6000.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.412324	-3.165230e-03	NaN	NaN
164	0.100	6000.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.105436	9.344041e-01	NaN	NaN
165	0.100	20000.0	0.100	0.0500	0.000000	76.0	48.0	32.0	403.0	0.108646	9.303500e-01	16.0	0.0
166	0.100	6000.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.103792	9.364344e-01	NaN	NaN
167	0.100	20000.0	0.100	0.0500	0.000000	76.0	48.0	32.0	403.0	0.106511	9.330597e-01	16.0	0.0
168	0.100	25000.0	0.100	0.0500	0.000000	128.0	64.0	32.0	403.0	0.411699	-1.253480e-04	16.0	0.0
169	0.100	25000.0	0.100	0.1000	0.000000	256.0	128.0	32.0	403.0	0.411756	-4.026479e-04	8.0	0.0
170	0.050	25000.0	0.100	0.0500	0.000000	256.0	128.0	16.0	403.0	0.112563	9.252368e-01	4.0	0.0
171	0.050	25000.0	0.100	0.0000	0.000000	190.0	90.0	30.0	403.0	0.411974	-1.465819e-03	3.0	0.0
172	0.100	8000.0	0.100	0.0000	0.000000	16.0	8.0	4.0	403.0	0.103529	9.367565e-01	2.0	0.0
173	0.100	15000.0	0.100	NaN	NaN	16.0	NaN	NaN	403.0	0.106009	9.336896e-01	NaN	NaN
174	0.100	15000.0	0.000	NaN	NaN	1.0	NaN	NaN	403.0	0.120667	9.140850e-01	NaN	NaN
175	0.100	15000.0	0.000	NaN	NaN	2.0	NaN	NaN	403.0	0.116812	9.194859e-01	NaN	NaN
176	0.100	35000.0	NaN	NaN	NaN	NaN	NaN	NaN	403.0	0.138819	8.862908e-01	NaN	NaN
177	0.050	25000.0	0.100	0.0100	0.000000	180.0	90.0	30.0	403.0	0.411965	-1.419538e-03	6.0	0.0
178	0.100	20000.0	0.100	0.0500	0.000000	76.0	48.0	32.0	403.0	0.110364	9.281301e-01	16.0	0.0

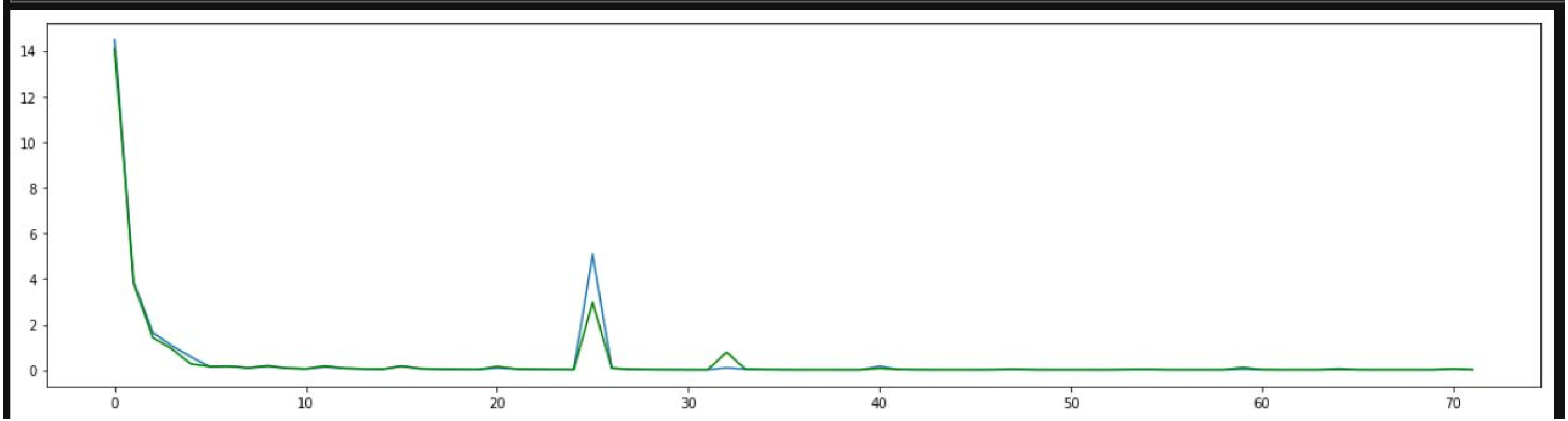
Observation and discovery :

- In the above parameter we can see that index 44 shows that for .001 learning parameter the model does not predict anything so I have changed it slowly and finally What I have found that learning parameter .1 and .05 provides the best results.
- Beta1, Beta2, Beta3, Beta4 represents the regularization parameter for hidden layer 1 ,2 ,3 and 4. Sometimes in the above table we can see that hidden layer 2,3,4 is 0 or NaN but there is some value for beta 2,3,4 that means the layer is actually off so those values actually means nothing.
- For 3 layer model when beta1, beta2, beta3 is .005, model shows significant amount of improvement while learning rate is .1 or .05 . But when learning rate is .1 and beta1=.1 , beta2=0, beta3=0 then the model performs even better most of the time and it also takes less epochs to train for the best validation accuracy
- From index 63 to 69 I have tried to use 200 , 100 , 30 neurons because the data have 403 features and its a common practice to use half amount of the neuron in the first hidden layer and this strategy does not work good enough but with my selected parameter it improved a little bit. I have used 16-8-4 combination of neuron because of this common practice. for our case 16 neuron in the first layer provided better accuracy and adding 8 and 4 in the next 2 layer improved the stability of the model and now it gives good validation accuracy after 2000 epoch and the best validation accuracy remains between the epoch range of 2000-2500 , 3300-3600 or 5000-5400 .
- From index 70 to 78 we can see that single neuron with single hidden layer performs well according to the plan stated in the target section. Then I have increased neurons and the learning curve for them is in the following block. Where y axis shows rmse and x axis shows i and i*50 represents the epoch no. Again blue curve is for training accuracy and green for validation accuracy
- In the table index 169 and 155 the model is exactly same with same parameter but one of them is providing .123 and other is providing .41 and that shows how inconsistent model become when we increase the neuron of the first hidden layer.
- We can see that even after adding another layer ANN does not perform well when we are increasing neurons in the first layer. The reason behind it is that this type of regression problem usually do well with logistic regression. By increasing neurons we cant do much improvement and all we need to do is properly regularize small amount of neurons so that they can perform well.

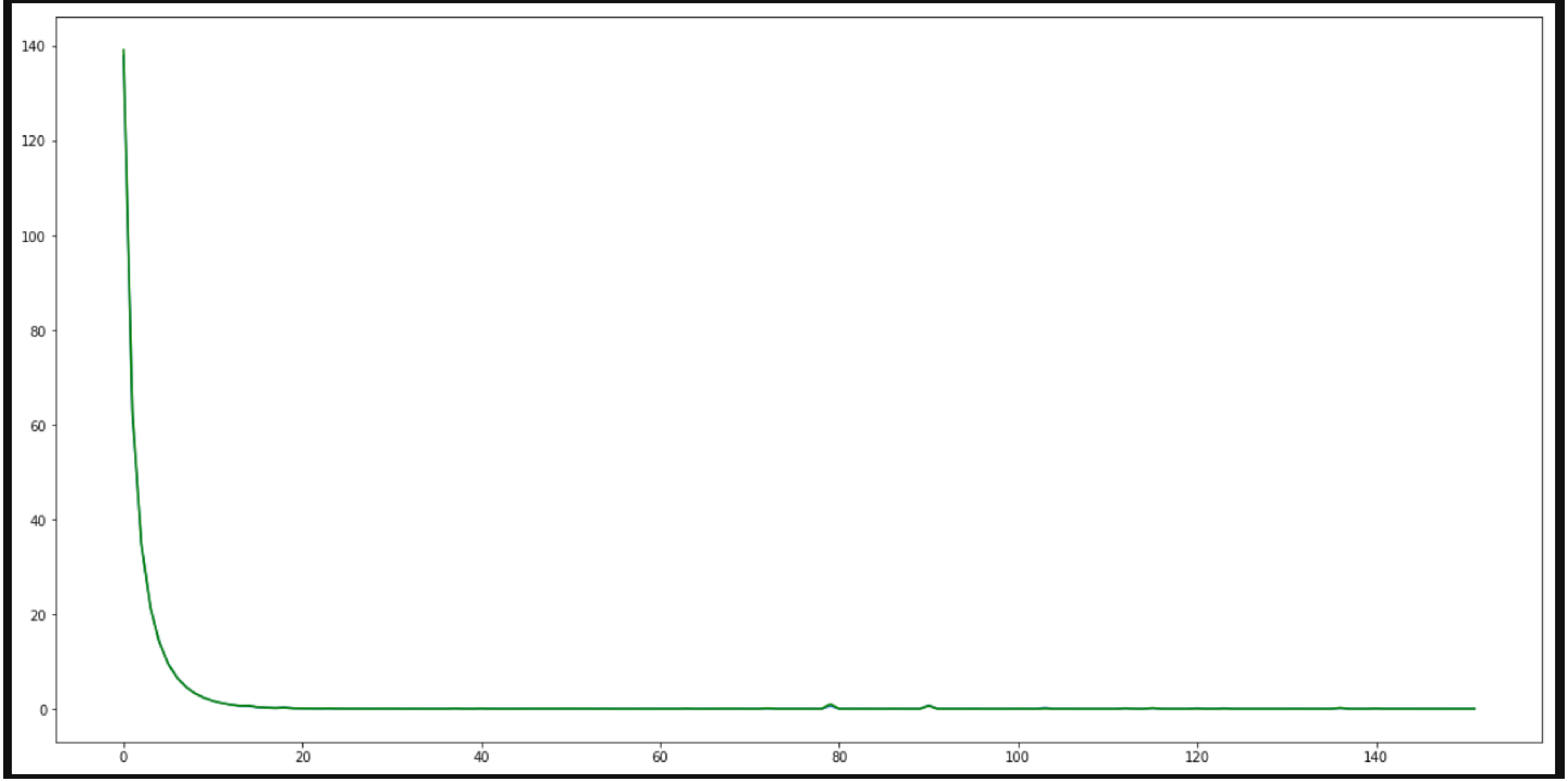
For single neuron learning rate



For 4 neuron learning curve



For 16 neuron learning curve



For 32 neuron learning curve

Ensemble

I am using bagging method for this section. Usually in this technique we add different models results and average them. But instead of averaging I am taking different fraction from different models result. Finally making sure that it sums up to 1.

I have tried different combinations of ensemble learning to improve performance. Kaggle has a certain limitation on uploading submission files. So what I have tried is that before submitting it to kaggle, I have made 80-20 split. I made prediction on the 20% data. Then I have tried ensemble learning so that before submission I can confirm which combination might work well.

In [304]:
```
if submit :
    pred_df = pd.read_csv("diffrent_pred_results.csv")
else:
    pred_df = pd.read_csv("pred_results.csv")


if not submit:
    pd.set_option('display.max_colwidth', -1)
    pred_df = pd.DataFrame(prediction_dict)
    pred_df.to_csv("pred_results.csv", encoding='utf-8',index=False)

else:
    pd.set_option('display.max_colwidth', -1)
    pred_df = pd.DataFrame(submit_prediction_dict)
    pred_df.to_csv("diffrent_pred_results.csv", encoding='utf-8',index=False)

pd.DataFrame(pred_df.columns)
```

Out[304]:

	0
0	Random Forest Regressor
1	DecisionTree
2	Xgboost
3	Lasso
4	ANN_base_lr0.1_beta0.1-0.0-0.0-None_hidden16-8-4-None
5	ANN_lr0.1_beta0.1-0.05-0.0-0.0_hidden76-48-32-16
6	ANN_lr0.05_beta0.005-0.1-0.05-0.0_hidden8-32-16-8
7	ANN_lr0.05_beta0.1-0.0-0.0-0.0_hidden16-8-4-2
8	ANN_lr0.1_beta0.1-None-None-None_hidden16-None-None-None
9	ANN_lr0.1_beta0-None-None-None_hidden4-None-None-None
10	ANN_lr0.1_beta0-None-None-None_hidden2-None-None-None

Naming explanation of above table

Name	learning rate	beta1	beta 2	beta 3	beta 4	hidden layer 1	hidden layer 2	hidden layer 3	hidden layer 4
ANN_base_lr0.1_beta0.1-0.0-0.0-None_hidden16-8-4-None	0.1	0.1	0.0	0.0	None	16	8	4	None
ANN_lr0.05_beta0.005-0.1-0.05-0.0_hidden8-32-16-8	0.05	0.005	0.1	0.05	0	8	32	16	6
ANN_lr0.05_beta0.1-0.0-0.0-0.0_hidden16-8-4-2	.05	0.1	0.0	0.0	0.0	16	8	4	2
ANN_lr0.1_beta0-None-None-None_hidden2-None-None-None	0.1	0	None	None	None	2	None	None	None
ANN_lr0.1_beta0.1-0.05-0.0-0.0_hidden76-48-32-16	0.1	.1	0.05	0.0	0.0	76	48	32	16
ANN_lr0.1_beta0.1-None-None-None_hidden16-None-None-None	0.1	0.1	None	None	None	16	None	None	None
ANN_lr0.1_beta0-None-None-None_hidden4-None-None-None	0.1	0	None	None	None	4	None	None	None

Ensemble Combination 1

In [317]:
```
# pred_df[pred_df.columns[[1,3,5]]] * [1,2,30]

print('Using  ' , pred_df.columns[[4,3,2]].values)
```

```
Using   ['ANN_base_lr0.1_beta0.1-0.0-0.0-None_hidden16-8-4-None' 'Lasso' 'Xgboost']
```

In [306]:
```
prediction = pred_df[pred_df.columns[[4,3,2]]] * [.4,.2,.4]
prediction = prediction.sum(axis = 1)

if not submit:
    test_rmse_score, test_r2_score = accuracy(y_test, prediction)

    print('ann root mean absolute error: ', test_rmse_score)
    print('accuracy score: ', test_r2_score  )
```

```
ann root mean absolute error:  0.1017685955982393
accuracy score:  0.9388884858512507
```

Kaggle score

output.csv 0.12231
21 hours ago by navid
Using ['ANN_base_lr0.1_beta0.1-0.0-0.0-None_hidden16-8-4-None' 'Lasso' 'Xgboost'] * [.4,.2,.4]

output.csv 0.12297
21 hours ago by navid
Using ['ANN_base_lr0.1_beta0.1-0.0-0.0-None_hidden16-8-4-None' 'Lasso' 'Xgboost'] * [.4,.2,.4]

In [307]:
```
prediction = pred_df[pred_df.columns[[4,3,2]]] * [.4,.3,.3]
prediction = prediction.sum(axis = 1)

if not submit:
    test_rmse_score, test_r2_score = accuracy(y_test, prediction)

    print('ann root mean absolute error: ', test_rmse_score)
    print('accuracy score: ', test_r2_score  )
```

```
ann root mean absolute error:  0.1017812588346656
accuracy score:  0.9388732764893758
```

Ensemble Combination 2

In [308]:
```
# pred_df[pred_df.columns[[1,3,5]]] * [1,2,30]

print('Using  ' , pred_df.columns[[2,4,5,6,7]].values)
```

```
Using   ['Xgboost' 'ANN_base_lr0.1_beta0.1-0.0-0.0-None_hidden16-8-4-None'
 'ANN_lr0.1_beta0.1-0.05-0.0-0.0_hidden76-48-32-16'
 'ANN_lr0.05_beta0.005-0.1-0.05-0.0_hidden8-32-16-8'
 'ANN_lr0.05_beta0.1-0.0-0.0-0.0_hidden16-8-4-2']
```

In [320]:
```
prediction = pred_df[pred_df.columns[[2,4,5,6,7]]] * [.25,.2,.2 ,.15 , .2]
prediction = prediction.sum(axis = 1)

if not submit:
    test_rmse_score, test_r2_score = accuracy(y_test, prediction)

    print('ann root mean absolute error: ', test_rmse_score)
    print('accuracy score: ', test_r2_score  )
```

```
ann root mean absolute error:  0.10272611129768247
accuracy score:  0.9377331075112767
```

Kaggle score

output.csv
21 hours ago by navid
Using ['Xgboost' 'ANN_base_lr0.1_beta0.1-0.0-0.0-None_hidden16-8-4-None' 'ANN_lr0.1_beta0.1-0.05-0.0-0.0_hidden76-48-32-16' 'ANN_lr0.05_beta0.005-0.1-0.05-0.0_hidden8-32-16-8' 'ANN_lr0.05_beta0.1-0.0-0.0-0.0_hidden16-8-4-2'] * [.25,.2,.2 ,.15 , .2]

0.12319

Ensemble Combination 3

In [310]:
```
print('Using  ' , pred_df.columns[[0,4,5,6,7]].values)
```

```
Using   ['Random Forest Regressor'
 'ANN_base_lr0.1_beta0.1-0.0-0.0-None_hidden16-8-4-None'
 'ANN_lr0.1_beta0.1-0.05-0.0-0.0_hidden76-48-32-16'
 'ANN_lr0.05_beta0.005-0.1-0.05-0.0_hidden8-32-16-8'
 'ANN_lr0.05_beta0.1-0.0-0.0-0.0_hidden16-8-4-2']
```

In [311]:
```
prediction = pred_df[pred_df.columns[[0,4,5,6,7]]] * [.25,.2,.2 ,.15 , .2]
prediction = prediction.sum(axis = 1)

if not submit:
    test_rmse_score, test_r2_score = accuracy(y_test, prediction)

    print('ann root mean absolute error: ', test_rmse_score)
    print('accuracy score: ', test_r2_score  )
```

```
ann root mean absolute error:  0.10392040423793271
accuracy score:  0.9362768645933334
```

Kaggle score

output.csv 0.12319

21 hours ago by navid

Using ['Random Forest Regressor' 'ANN_base_lr0.1_beta0.1-0.0-0.0-None_hidden16-8-4-None' 'ANN_lr0.1_beta0.1-0.05-0.0-0.0_hidden76-48-32-16' 'ANN_lr0.05_beta0.005-0.1-0.05-0.0_hidden8-32-16-8' 'ANN_lr0.05_beta0.1-0.0-0.0-0.0_hidden16-8-4-2'] * [.25,.2,.2 ,.15 , .2]

Ensemble Combination 4

In [316]:
```
print('Using  ' , pred_df.columns[[4,5,6,7,0,2,3]].values)
```

```
Using   ['ANN_base_lr0.1_beta0.1-0.0-0.0-None_hidden16-8-4-None'
 'ANN_lr0.1_beta0.1-0.05-0.0-0.0_hidden76-48-32-16'
 'ANN_lr0.05_beta0.005-0.1-0.05-0.0_hidden8-32-16-8'
 'ANN_lr0.05_beta0.1-0.0-0.0-0.0_hidden16-8-4-2' 'Random Forest Regressor'
 'Xgboost' 'Lasso']
```

In [315]:
```
prediction = pred_df[pred_df.columns[[4,5,6,7,0,2,3]]] * [.15,.1,.1,.05,.0,.2,.4]
prediction = prediction.sum(axis = 1)

if not submit:
    test_rmse_score, test_r2_score = accuracy(y_test, prediction)

    print('ann root mean absolute error: ', test_rmse_score)
    print('accuracy score: ', test_r2_score  )
```

```
ann root mean absolute error:  0.10192734103655354
accuracy score:  0.9386976855299631
```

Kaggle score

output.csv 0.12192

18 hours ago by Navid

Using ['ANN_base_lr0.1_beta0.1-0.0-0.0-None_hidden16-8-4-None' 'ANN_lr0.1_beta0.1-0.05-0.0-0.0_hidden76-48-32-16' 'ANN_lr0.05_beta0.005-0.1-0.05-0.0_hidden8-32-16-8' 'ANN_lr0.05_beta0.1-0.0-0.0-0.0_hidden16-8-4-2' 'Random Forest Regressor' 'Xgboost' 'Lasso'] * [.15,.1,.1,.05,.0,.2,.4]

Ensemble combination 4 provides the best score which is 0.12192. Currently combination 4 is showing that rmse value is .1019 and there is a better value present in combination 1 which is 0.1017. The reason behind the difference is ANN does not perform exactly same each time. That means if I currently submit with Combination 1 I might get better results than Combination 4 .

In Combination 4 used models with their parameters:

Name	learning rate	beta1	beta 2	beta 3	beta 4	hidden layer 1	hidden layer 2	hidden layer 3	hidden layer 4	Fraction taken
ANN_base_lr0.1_beta0.1-0.0-0.0-None_hidden16-8-4-None	0.1	0.1	0.0	0.0	None	16	8	4	None	.15
ANN_lr0.05_beta0.005-0.1-0.05-0.0_hidden8-32-16-8	0.05	0.005	0.1	0.05	0	8	32	16	6	.1
ANN_lr0.05_beta0.1-0.0-0.0-0.0_hidden16-8-4-2	.05	0.1	0.0	0.0	0.0	16	8	4	2	.05
ANN_lr0.1_beta0.1-0.05-0.0-0.0_hidden76-48-32-16	0.1	.1	0.05	0.0	0.0	76	48	32	16	.1
Xgboost	0.05	Not applicable	Not applicable	Not applicable	Not applicable	Not applicable	Not applicable	NonNot applicablee	Not applicable	.2
Lasso	alpha = 5e-4	Not applicable	Not applicable	Not applicable	Not applicable	Not applicable	Not applicable	NonNot applicablee	Not applicable	.4

Obesrvation

In the learning curve graph if the minimum of training and validation is close to each other then its good to use that model. Again if training minimum and validation minimum is no where near each other then using them does not help much most of the case. When both of them are close we can use the epoch no of the train_min loss as val_min loss epoch no and then we can train over all the dataset without depending on the epoch number. The model does not give same result in same epoch every time. This is the main reason behind removing the epoch dependency.

Prepare Submission File

To use this section please uncomment the last line of split data section and comment accuracy section.

In [51]:
```
use_ensemble = True
#if want to use given test data
if submit:
    X_test = test_processed
    if not use_ensemble:
        with tf.Session() as sess:
            # Restore variables from disk.
            saver.restore(sess, "model/model.ckpt")
            print("Model restored.")

            # Check the values of the variables
            pred =  sess.run(model, feed_dict={X_tf: X_test})
            prediction = pred.squeeze()

    prediction = np.exp(prediction.values)

    pred_out_df = pd.DataFrame(prediction, index=test["Id"], columns=["SalePrice"])
    pred_out_df.to_csv('output.csv', header=True, index_label='Id')
```

Conclusion & Kaggle score Discussion:

My target of this report was to improve ANN model and show how well it can perform with ANN model. In the beginning of the report I have build a ANN model that performs better than any other single ANN model. I have performed cross validation on that model and that model scored 0.12324 in kaggle. Then I have showed some other models that performs well but can't beat the score .12324 . Then I have explained why some models with certain parameter works well. After that I showed a table where different models performance is listed and added my analysis and observation. Then I have have Showed four combination of Ensemble and their kaggle score is also attached with them. In the 4th combination of Ensemble method I have found the best kaggle score which is 0.12192. This is the overall best score and achived through combining 4 ann models, xgboost and lasso.

Reference

xgboost:

https://www.kaggle.com/dansbecker/xgboost (https://www.kaggle.com/dansbecker/xgboost)

https://medium.com/@gabrieltseng/gradient-boosting-and-xgboost-c306c1bcfaf5 (https://medium.com/@gabrieltseng/gradient-boosting-and-xgboost-c306c1bcfaf5)

regression + graph :

https://www.kaggle.com/janiobachmann/predicting-house-prices-regression-techniques (https://www.kaggle.com/janiobachmann/predicting-house-prices-regression-techniques)

Selecting and Filtering Data

https://www.kaggle.com/dansbecker/selecting-and-filtering-in-pandas (https://www.kaggle.com/dansbecker/selecting-and-filtering-in-pandas)

Handling Missing Values

https://www.kaggle.com/dansbecker/handling-missing-values (https://www.kaggle.com/dansbecker/handling-missing-values)

why use conditional probability coding

https://medium.com/airbnb-engineering/designing-machine-learning-models-7d0048249e69 (https://medium.com/airbnb-engineering/designing-machine-learning-models-7d0048249e69)

one hot encoding

https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f (https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f)

https://medium.com/@rajatgupta310198/getting-started-with-neural-network-for-regression-and-tensorflow-58ad3bd75223 (https://medium.com/@rajatgupta310198/getting-started-with-neural-network-for-regression-and-tensorflow-58ad3bd75223)

class example

https://colab.research.google.com/drive/1MExQ52bvHSPaUrGe8RvHZifvE6K6a0qh?fbclid=IwAR2EUWi4q6_q0mFbXQwGh4GNgB2Ex_WpP3K0L12182PdzszWSsEfzHf0REo#forceEdit=true&offline=true&sandboxMode=true&scrollTo=-Rh3-Vt9Nev9 (https://colab.research.google.com/drive/1MExQ52bvHSPaUrGe8RvHZifvE6K6a0qh?fbclid=IwAR2EUWi4q6_q0mFbXQwGh4GNgB2Ex_WpP3K0L12182PdzszWSsEfzHf0REo#forceEdit=true&offline=true&sandboxMode=true&scrollTo=-Rh3-Vt9Nev9)

Why cross validation

https://towardsdatascience.com/5-reasons-why-you-should-use-cross-validation-in-your-data-science-project-8163311a1e79 (https://towardsdatascience.com/5-reasons-why-you-should-use-cross-validation-in-your-data-science-project-8163311a1e79)

Decision Tree - Regression

https://www.saedsayad.com/decision_tree_reg.htm (https://www.saedsayad.com/decision_tree_reg.htm)

Some more

https://www.kaggle.com/klyusba/house-prices-advanced-regression-techniques/lasso-model-for-regression-problem/notebook (https://www.kaggle.com/klyusba/house-prices-advanced-regression-techniques/lasso-model-for-regression-problem/notebook)

https://www.kaggle.com/juliencs/house-prices-advanced-regression-techniques/a-study-on-regression-applied-to-the-ames-dataset/ (https://www.kaggle.com/juliencs/house-prices-advanced-regression-techniques/a-study-on-regression-applied-to-the-ames-dataset/)

https://www.kaggle.com/apapiu/house-prices-advanced-regression-techniques/regularized-linear-models (https://www.kaggle.com/apapiu/house-prices-advanced-regression-techniques/regularized-linear-models)

https://www.kaggle.com/juliencs/a-study-on-regression-applied-to-the-ames-dataset (https://www.kaggle.com/juliencs/a-study-on-regression-applied-to-the-ames-dataset)

For descriptive section

I have inspired form Ian Goodfellows book and used his way of explanation to explain my choice. His book can be found here: https://www.deeplearningbook.org/ (https://www.deeplearningbook.org/)