House Value Prediction using 1-Dimensional Convolutional Neural Network

Sahand Asri
Department of Computer Science
Lakehead University
Thunder Bay, Canada
sasri@lakeheadu.ca

***Abstract*—In this assignment, we are working on California Housing dataset. Our objective is to predict the median house value for households within a block in California which is measured in US dollars. For this task, we need an artificial intelligence model which gathers the features and outputs from dataset and learns the connection between them to anticipate the median value of house that was not given before to the model. In this paper, we are using Convolutional Neural Network to perform non-linear regression. To obtain the best results, different parameters and architectures are built and evaluated.**

***Index Terms*—California Housing, Convolutional Neural Network, Scikit-learn**

I. Introduction

Convolutional Neural network (CNN) is one of the most popular Deep Neural Network (DNN) architectures based on natural visual perception of the living animals [1].

In [2], Hubel and Wiesel explained that some cells in the visual cortex of animals such as cats and monkies are responsible for detecting light in perceptive fields. Based on this discovery, Kunihiko Fukushima offered a self-organizing neural network model named neocognitron [3]. The proposed model is a neural network attempting to mimic the hierarchical and compositional characteristics of the visual cortex discovered by the previous paper. Neocognition is often regarded as the predecessor of convolutional neural networks. At that time, training phased, which consists of weight optimization and backpropagation, has not been applied to neural networks. Therefore, there was no method to optimize the neocognition model to use in different applications.

Later on, Alex Waibel introduced Time Delay Neural Network (TDNN) in 1989 [4]. This neural network was the first model that applied global weight optimization on the neocognition neural network model. In 1998, LeCun published a paper titled Gradient-based learning applied to document recognition. In this paper, they established modern CNN by developing a multi-layer artificial intelligence neural network named LeNet-5. They used this architecture for handwritten digit classification. This model was the first convolutional neural network that performed on a real industrial application. The strength of this model was that it accepted raw pixels (inputs) without any preprocessing steps to recognize the visual patterns of handwritten digits.

After that, various architectures and methods were offered for a deep CNN. The most notable one was proposed by Krizhevsky et al., which was developed for an image classification contest on ImageNet in 2010 [5]. This model is often referred to as AlexNet. AlexNet was using a deep CNN architecture with significant enhancements upon the previous architectures. AlexNet was designed to classify 1.2 million images into 1000 different classes. For top-1 and top-5, they achieved the error rates of 37.5% and 17% [5]. After the successful attempt of AlexNet, many CNN architectures were offered to improve the performance more. VGGNet [6], GoogleNet [7], ResNet [8] are some of the well-known ones.

In the following sections, we explain the basic components of a Convolutional Neural Network. After that, we introduce the dataset which we are working on. Then, we present our proposed CNN architecture for a non-linear regression task. In the end, we discuss the performance of our proposed model with different parameters.

II. CNN component

There are various architectures for CNN systems. However, there are some similarities between them in the primary elements. For instance, in LeNet-5, there are mainly three types of layers which are convolutional layer, pooling layer, and fully-connected layer.

A. Convolutional Layer

The convolutional layer is the first layer in a CNN architecture. This layer receives the raw inputs from the user and aims to extract some features from them. A Convolutional layer may contain several convolutional kernels. These convolutional kernels are used for computing the feature maps. In this layer, first, inputs convolve with convolutional kernels. Then, the convolved results go to the non-linear activation functions. Typical activation functions are ReLU, sigmoid and tanh.

B. Pooling Layer

The pooling layer usually goes between two convolutional layers. The task of the pooling layer is to reduce the resolution of the feature maps, generated by the previous convolutional layer. By decreasing the resolution of the feature maps, CNN will achieve shift-invariance. In other words, by applying pooling layers, convolutional layers in CNN could better extract different features.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY
5	-122.25	37.85	52.0	919.0	213.0	413.0	193.0	4.0368	269700.0	NEAR BAY
6	-122.25	37.84	52.0	2535.0	489.0	1094.0	514.0	3.6591	299200.0	NEAR BAY
7	-122.25	37.84	52.0	3104.0	687.0	1157.0	647.0	3.1200	241400.0	NEAR BAY
8	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.0804	226700.0	NEAR BAY
9	-122.25	37.84	52.0	3549.0	707.0	1551.0	714.0	3.6912	261100.0	NEAR BAY

Fig. 1. California Housing dataset properties

C. Fully-Connected Layer

After several convolutional and pooling layers, we may face one or more fully-connected layers. Now, we extract and gather the high-level features which we need for this layer. Fully-connected layer connects the previous nodes to every one of the neurons in the current layer.

D. Output Layer

The last layer of a classic CNN model is the output layer. Output layers could be different based on the system's application. For instance, for a classification problem, output layers mostly perform softmax operator. In addition to that, the output layer may use SVM. The loss of CNN is calculated in this stage. There are many types of loss functions used to calculate the performance of the system's prediction. CNN finds the best and optimum parameters by reducing the loss calculated by the loss function in the last layer. Gradient descent is often used for network optimization.

III. Dataset

California Housing is a dataset that was built on the year of 1990 based on California census data. This dataset emerged in a paper titled Sparse Spatial Autoregressions, published in Statistics and Probability Letters journal. California Housing contains block groups of people living in various neighbourhoods of California. Each entry of the dataset represents a single block group from 600 to 3,000 people. On average, a block group consists of 1425.5 individuals. As you can see in the Figure 1, there are ten properties for each entry of the dataset. Nine properties out of ten are numerical attributes, and one is categorical.

IV. Proposed Model

We implement a one-dimensional Convolutional (Conv1D) Neural Network. Our objective is to design a CNN model which predicts the median house value of an entry. The inputs of this model are features of the California Housing dataset exclude median_house_value and ocean_proximity, since the former is the output value, and the latter is a categorical attribute.

In this project, to achieve the best performance, we designed different architectures for our CNN model. For example, we tested several numbers of layers, neurons, and activation functions.

In addition to that, we consider different parameters in the model, namely: batch_size, kernel_size, stride, padding, and epoch_num.

	Specs	Score
3	total_rooms	8.872994e+06
5	population	3.709531e+06
4	total_bedrooms	1.310306e+06
6	households	1.197789e+06
2	housing_median_age	2.103298e+04

Fig. 2. The best five features, selected by SelectKBest

Also, to reach the best results, we performed some preprocessing steps, such as selecting the best features using sklearn's SelectKBest class. For this, we choose the best five features out of eight attributes based on chi-square (chi2) test measures. We need to mention that chi2 does not accept negative attributes.

Therefore, we shift the longitude and latitude by adding the minimum value to every cell. In Figure 2, the best five features are displayed, which are: total_rooms, population, total_bedrooms, households, housing_median_age.

Also, to optimize the learning process, we use step learning rate decay, where we divide the learning rate by two after passing a particular number of epochs every time. For instance, the learning rate is divided by 2 every time the epoch number is divisible by 2000. This process continues until the learning stops. The source code for simple step learning rate decay is displayed in Figure 4.

V. Experimental Analysis

Our model was evaluated based on the test results in terms of MSE and R^2. As we mentioned, different architectures, methods, and optimization methods were used for this problem. The evaluation metrics were fluctuating even for the

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

min_latitude = x_train["latitude"].min()
min_longitude = x_train["longitude"].min()
x_train["latitude"] = x_train["latitude"]
    + abs(min_latitude)
x_train["longitude"] = x_train["longitude"]
    + abs(min_longitude)

# apply SelectKBest to extract top 8 best features
bestfeatures = SelectKBest(score_func=chi2, k=8)
fit = bestfeatures.fit(x_train,y_train)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x_train.columns)
# concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
#naming the dataframe columns
featureScores.columns = ['Specs','Score']

#print 5 best features
print(featureScores.nlargest(5,'Score'))
```

Fig. 3. Source code for selecting best k features using SelectKBest

```
losses = []
r2_scores = []
for cnt, epoch in enumerate(range(epochs)):
  avg_loss, avg_r2_score = model_loss(model,
    loader, train=True, optimizer=optimizer)
  losses.append(avg_loss)
  r2_scores.append(avg_r2_score)
  if(cnt % 2000 == 1):
    lr /= 2
```

Fig. 4. Source code for learning rate decay

same model and methods, after running for multiple times. Therefore, we realised that testing the performance of the model once or twice is not a perfect way to evaluate the efficiency of the model. For instance, for one of the tested models, for average R^2 score we obtained the value of 0.43. However, the second time we only reached 0.19.

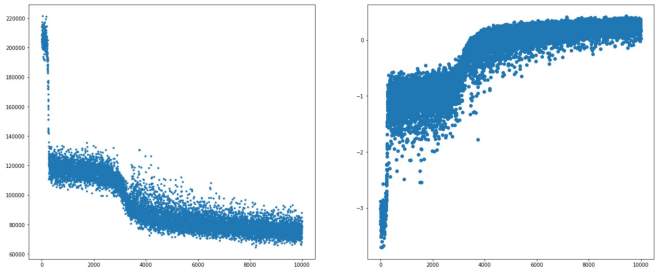

Fig. 5. MSE Loss function on left, R^2 score on right

```
The model's L1 loss is: 67314.890625
The model's R^2 score is: 0.4086396320663652
```

Fig. 6. MSE Loss and R^2 Score

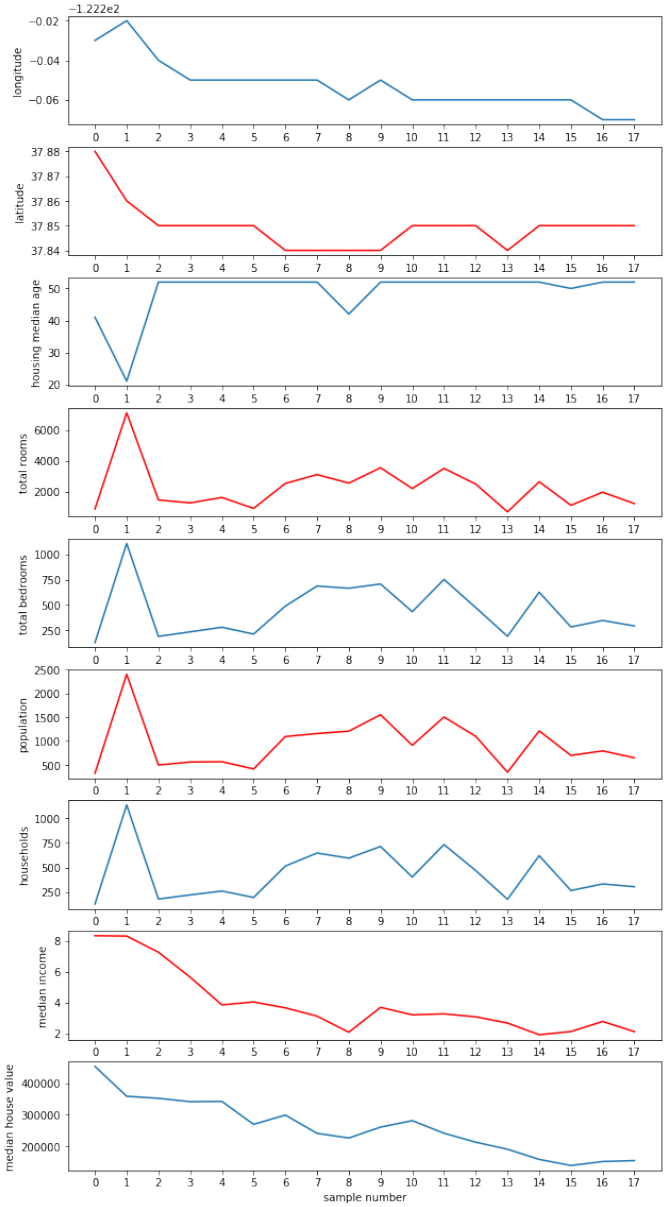

Fig. 7. California Housing features for the first 18 entries

REFERENCES

[1] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai *et al.*, "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354–377, 2018.

[2] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[3] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[4] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE transactions on acoustics, speech, and signal processing*, vol. 37, no. 3, pp. 328–339, 1989.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.