Enhancing Data Resolution Using Deep Learning

Rusmia Sattar[†]

[†] *Department of Mathematics and Statistics, Memorial University of Newfoundland, St. John's (NL) A1C 5S7, Canada*

E-mail: rsattar@mun.ca

Abstract
With a primary focus on the use of a feedforward neural network to approximate functions that allow the conversion of low-resolution data into high-resolution data, this research explores the field of deep learning. This study investigates the network's ability to close the resolution gap, with special emphasis on how it improves data representation. Focusing on the functionality and adaptability of this neural network design through careful testing and research, we are shedding light on its possible uses in situations needing data upscaling.

Keywords: Deep Learning, Tensorflow, Feedforward Neural network, Data resolution

1 Introduction

In an era defined by exponential technological advancements, the field of artificial intelligence has emerged as a formidable force. It is reshaping the way we interact with the digital world and unlocking novel capabilities that were once deemed the stuff of science fiction. One of the most outstanding developments in this revolutionary environment is Deep Learning.[1] Deep learning is a branch of machine learning that draws inspiration from the complex operations of the human brain to develop artificial neural networks that are capable of amazing feats.

The history of deep learning is intertwined with those of machine learning and artificial intelligence (AI), which are two related but distinct fields. Deep Learning has been a revolutionary force in recent years, spurring innovation in a variety of fields and altering the way we approach challenging AI challenges. When it comes to tasks like interpreting natural language, image recognition, and autonomous decision-making, neural networks have unmatched powers.[2]

In the realm of modern data analysis and artificial intelligence, the use of Deep Learning techniques has revolutionized our ability to process and interpret complex data patterns. The ability of Deep Learning and to address the difficult issue of improving data resolution, is explored in this project. We explore function approximation and data transformation by utilizing the TensorFlow framework's capabilities.[3]

To embark on this computational journey, we first generate a low-resolution dataset, which serves as our reference. Subsequently, we create high-resolution datasets by subsampling the low-resolution data at varying intervals. These low-resolution datasets mirror scenarios where data is limited or imprecise, a common occurrence in real-world applications. This project's implementation of a feedforward neural network forms its core. We want to understand the neural network's convergence behavior by training it for 100 epochs and then we want to assess how well the network approximates the high-resolution function [4].

In the end, we want to show how Feedforward Neural Networks, in particular, may improve data resolution and approximate functions. We will have a better knowledge of the network's

capabilities and its potential for real-world applications by the end of this project.

2 Methods

In this project, a deep learning resolution experiment is conducted to bridge the gap between low-resolution and high-resolution data using a feedforward neural network implemented with TensorFlow's Keras Sequential API [1]. Here we implemented a code using Google CoLab to conduct this data resolution activity. Improving the resolution of data can take many forms simple to complex. Here we just use a simple function to conduct the experiment. The experiment begins by defining a complex mathematical function, f(x), representing the high-resolution data to be approximated.

$$f(x) = (1-x)sin(2x) + (1-x)^2 sin(10x) \tag{1}$$

This function is difficult to adequately model as it combines oscillatory and polynomial components. A low-resolution dataset is generated by evenly sampling x values between 0 and 1 that simulate settings with limited or inaccurate data.//

```
# Generating low-resolution data using the original function with one subsampling factor
x_low_res = np.linspace(0, 1, i)
y_low_res = f(x_low_res)
# Reshaping X and y to match the length of x_low_res
X = x_low_res.reshape(-1,1)
y = y_low_res.reshape(-1,1)

# Generating the high-resolution dataset
X_high_res = np.linspace(0, 1, 101)
Y_high_res = f(X_high_res)
```

Subsamples of the low-resolution data are then taken at different intervals to build high-resolution datasets using Tensorflow[5]. Additionally, ReLU and Linear activation functions from Keras Sequential API have been designed and modified as shown below:

```
    model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(1,)),
    keras.layers.Dense(1, activation='linear')  # Output layer size is 1
])
```

The model is trained for 100 epochs and its performance is evaluated using mean squared error (MSE).

```
 # Compiling the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Training the model
hist = model.fit(X, y, epochs=100, verbose=0)  # Use X and y for training
```

The high-resolution function, low-resolution data, and model predictions are displayed using visualization tools. This visual depiction allows us to assess how well the neural network executes the data processing.

3 Results

The outcomes of the deep learning resolution experiment were informative. As seemed to be predicted, the subsampling factor, which represented various levels of data resolution, considerably affected how well the neural network performed. To represent the complex patterns of the high-resolution function at lower subsampling factors was more difficult, where data was more limited and lower in resolution, which resulted in higher mean squared errors (MSE) portrayed in Figure 1.

On the other hand, the model performed better with greater subsampling factors, with lower MSE values suggesting more accurate approximations. Performance was also significantly influenced by the neural network architecture and activation algorithms.

Deeper networks with the right configurations tended to perform better than more straightforward ones. Overall, the experiment showed how deep learning, in particular feedforward neural networks, has the ability to improve data resolution and approximate complex functions. We used the tensor flow and Keras API tools to enhance the resolutions and compare them with the loss and prediction points. When working with data of different resolutions, these findings underline the significance of proper network design and data augmentation.

```
Subsample factor: 20
WARNING:tensorflow:6 out of the last 6 calls to <function M
1/1 [==============================] - 0s 64ms/step
Loss: 0.06805907189846039
Subsample factor: 10
1/1 [==============================] - 0s 57ms/step
Loss: 0.08441399037837982
Subsample factor: 5
1/1 [==============================] - 0s 51ms/step
Loss: 0.05894386023283005
Subsample factor: 2
1/1 [==============================] - 0s 55ms/step
Loss: 1.808294882721384e-07
```

Figure 1

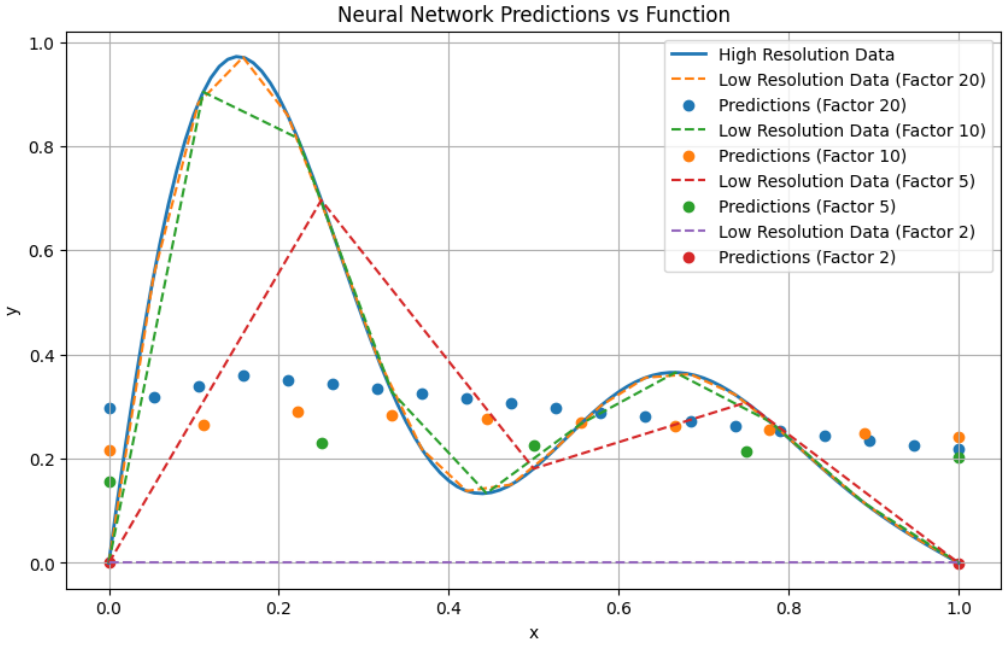

Figure 2

Here in Figure 2, we can see the outcomes derived from the "Neural Network Predictions vs Function" plot. The low-resolution data fluctuates a lot over time but the high-resolution data

first rises and then declines rapidly over time and then starts to rise again gradually. It also fluctuates a lot more than the prediction points.

4 Conclusion

In summary, our project investigated the capabilities of deep learning, specifically feedforward neural networks. We worked on training our model to increase data resolution and function approximation. We discovered that the resolution of the data, network architecture, and activation functions all influenced the neural network's performance. Despite the fact that the low-resolution data fluctuated, the model successfully converged from low-resolution data to high-resolution. This study demonstrates the revolutionary potential of deep learning in real-world applications with varying data quality. It highlights the significance of careful network architecture and data augmentation. It also highlights how artificial intelligence is altering our capabilities and expanding our data analysis perspectives.

However, the model has several drawbacks, such as a dependency on the quantity and quality of training data, a potential inability to handle complicated real-world data, a limited ability to generalize to different domains, a high demand for computational resources, and difficulties limiting overfitting. These drawbacks highlight the necessity of thoughtful consideration and modification when using the model in real-world settings.

In essence, this project underscores the transformative potential of deep learning, especially in improving data resolution and solving complex tasks, notably in the context of feedforward neural networks. It highlights the importance of careful network design and data augmentation in real-world applications where data quality varies. In an era of rapid technological developments, this project demonstrates the limitless possibilities of artificial intelligence, pushing the boundaries of what was previously science fiction and opening new possibilities for data analysis across multiple fields.

Acknowledgements

I gratefully acknowledge the cordial help of my respected Professor for this project. This is my first ever project report in technical writing using Latex and I had the valuable support of my peers. Additionally, I would like to declare that, I have used chatGPT to provide a draft for the introduction and conclusion of this report, which I have then reviewed and edited.

References

[1] Francois Chollet et al. Keras, 2015.

[2] Wenming Yang, Xuechen Zhang, Yapeng Tian, Wei Wang, Jing-Hao Xue, and Qingmin Liao. Deep learning for single image super-resolution: A brief review. *IEEE Transactions on Multimedia*, 21(12):3106–3121, 2019.

[3] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[4] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 2022.

[5] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

Appendix

```
#Rusmia Sattar
#Generating high resolution data from low resolution data using tensor flow and keras

import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras

#Defining the given function

def f(x):
    return (1 - x) * np.sin(2 * x) + (1 - x)**2 * np.sin(10 * x)


# Generating the high-resolution dataset
X_high_res = np.linspace(0, 1, 101)
Y_high_res = f(X_high_res)

points = [20, 10, 5, 2]

#Setiing the plot for the high resolution data
plt.figure(figsize=(10,6))
plt.plot(X_high_res, Y_high_res, label='High Resolution Data', linewidth=2)


for i in points:
    print(f"Subsample factor: {i}")

# Generating low-resolution data using the original function with one subsampling factor
    x_low_res = np.linspace(0, 1, i)
    y_low_res = f(x_low_res)
# Reshaping X and y to match the length of x_low_res
    X = x_low_res.reshape(-1,1)
    y = y_low_res.reshape(-1,1)

    model = keras.Sequential([
        keras.layers.Dense(64, activation='relu', input_shape=(1,)),
        keras.layers.Dense(1, activation='linear')  # Output layer size is 1
    ])

    # Compiling the model
    model.compile(optimizer='adam', loss='mean_squared_error')

    # Training the model
    hist = model.fit(X, y, epochs=100, verbose=0)  # Use X and y for training

    # Generating predictions
    y_pred = model.predict(X)

    # Plotting the results as needed
    print("Loss:", hist.history['loss'][-1])
    plt.plot(x_low_res, y_low_res, label=f'Low Resolution Data (Factor {i})', linestyle='dashed')
    plt.scatter(x_low_res, y_pred, marker='o', label=f'Predictions (Factor {i})')

plt.xlabel('x')
plt.ylabel('y')
plt.title('Neural Network Predictions vs Function')
plt.legend()
plt.grid(True)
plt.show()
```