Introduction to Data Upscaling using Deep Neural Networks

Saurav Singh Chandel[†]

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

E-mail: sschandel@mun.ca

Abstract:
This paper briefly introduces mathematical function upscaling techniques based on deep learning. These approaches employ Feed-Forward Neural Networks to elevate the resolution of low-resolution datasets, producing high-quality, intricate outcomes. The process encompasses the training of the model on a dataset comprising pairs of lower and relatively higher-resolution datasets, with considerations for data preprocessing, network architecture, loss functions, and optimization strategies. In this exploration, the paper also delves into an examination of the effects produced by different activation functions and varying numbers of neurons employed within the neural network architecture. Furthermore, the practical applicability of these techniques extends to real-world scenarios such as image upscaling.

Keywords: Machine learning, Upscaling, Neural Networks, ANNs, CNNs

1 Introduction

1.1 What is Deep Learning?

Before getting into technical specification, we can try to imagine it by considering the following case:
If we have a function *sin(x)* on a specified interval and we want to fit our linear function such that it completely overlaps the given function and also predict the function accurately outside the interval, then we can use Deep learning neural networks to get the desired results.
We can think of Deep learning as a set of techniques to manipulate multiple such linear functions and define a really complex piecewise function that gives us the desired output.

Deep learning is a subset of machine learning specializing in working with unstructured data with minimal user optimization on the front end. Deep learning tries to mimic the workings of the human brain by artificially implementing neurons. These neurons are usually aligned in layer structures where each layer has its own mathematical function for manipulating the input and sending the processed output to the next layer. Any neural network consists of at least 2 layers also called *Input layer* and *Output layer*.

Then the user can define multiple hidden layers with different numbers of nodes and different activation functions. Each layer will have the same activation function for all the neurons in it. All the neurons are connected to all the neurons of the adjacent layers. Each neuron has its *weights, biases, and an activation* function which are in the form of $\phi(\mathcal{W}\mathsf{x} + \mathcal{B})$. Where ϕ is the activation function, x is the input, $\mathcal{W}$ is the weight of the neuron, and B is the bias.

The working of a neural network can be divided into two steps. *Forward Propogation* is where the data is passed through a set of weights and biases. The very first forward propagation cycle chooses weights and biases randomly. The second step *Back Propogation* is where the neural network backtracks and optimizes weights and biases in order to minimize the error. [1] The model starts its training cycles which are also called *epochs*. After every cycle, with the help of user-specified loss functions and backpropagation, the weights and biases of all the neurons are changed to minimize the error in the final output. The manipulation of weights and biases affects the final output of the neural network. Their individual working is explained in further detail in the next section.

There are two main applications for Deep learning neural networks:

Classification: Neural networks are used to identify images, sentiment analysis or classify data in discrete classes. Eg: Character recognition

Estimation: Neural networks are used to estimate missing data and predict behaviors. Eg: Image upscaling

1.2 What makes a neural network?

1.2.1 Neurons

We can think of each neuron as a single mathematical function defined as $\phi(\mathcal{W}\mathsf{x} + \mathcal{B})$.
x is the input to the neuron from the previous neuron. The output from the previous neuron is used as the input for the next neuron.
$\mathcal{W}$ is the linear multiplier of the input and $\mathcal{B}$ is the offset of the expression before it is passed

into the activation function.
ϕ is the activation function defined for that layer. Activation functions normalize the inside expression and send their output to the next neuron. We will discuss more about different activation functions later.

A neuron is the smallest functional unit of a neural network; each neuron in a layer is connected to all the neurons in the previous and next layers. They work similarly to long chains whose weights and biases can be altered and optimized to get the desired results.

1.2.2 Layers

Layers are just a group of parallel neurons. The neurons in the same layer have the same activation function but they are not connected to each other. Layers can be thought of as breaking the big problem of predicting something with a neural network into small chunks where each layer handles a smaller part of the problem. We can experiment with different numbers, types, and sizes of layers to optimize our final result.

1.2.3 Activation Function

Activation functions are the wrapper functions around a neuron that take the weights and biases as parameters and give normalized output usually in the interval [0, 1]. There are several different types of activation functions such as ReLU, Sigmoid, etc.
Activation functions play a crucial part in the accuracy of the final output of the neural network. The choice of activation functions is really important and depends on the use case.

1.2.4 Loss Function

Loss functions are mathematical functions used during the back propagation part of deep learning which calculate the error of the calculated output from the expected output. They use a metric to define the accuracy of the prediction of the neural network. The goal is to minimize the loss function as much as possible to get the best possible results from the machine learning model.

1.2.5 Optimizers

Optimizers are algorithms or methods that are specifically designed to reduce the loss by altering the weights and biases of the neurons. One of the most commonly used optimizers is Gradient Descent which finds the negative slope of an n-dimensional object and the loss function acts as a guide for the optimizer to navigate and find the lowest error.

2 Methods

There are several different types of methods of training neural networks but we are going to discuss *Artifical Neural Networks (ANNs)* and *Convolution Neural Networks (CNNs)* which are some of the most commonly used methods to train neural networks.

2.1 Artifical Neural Network (ANN)

Artificial Neural Networks also known as *Feed-Forward* neural networks because the inputs are only processed in the forward direction. They are specifically known for their remarkable strength in approximating functions hence they are popularly also known as *Universal Function*

Approximators. They are capable of learning any non-linear function because of the activation functions. They work best with 1-dimensional vectors. We use many feed-forward networks to predict and upscale low-resolution dataset of a complex function to high resolution. They are not suited for *Image recognition* as that requires the 2-dimensional image to be converted into a 1-dimensional vector which significantly increases the process. Also, if we have a deep neural network with a large number of hidden layers, during backpropogation the Gradient vanishes or explodes therefore giving inconsistent results. [3]

2.2 Convolution Neural Network (CNN)

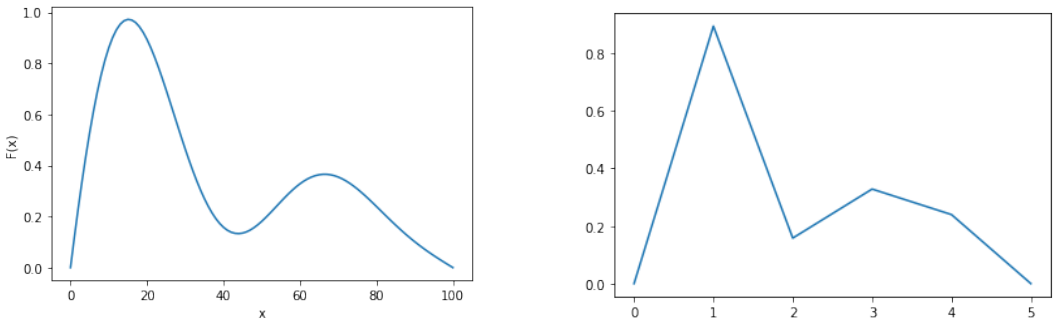
Convolutional Neural Networks are specially designed to process grid-based data such as images. They are quite similar to ANNs but one notable difference is that they are not always fully connected to the adjacent layers. They have a special *Convolutional layer* which determines the output of neurons connected to local regions of the input. They also have a *Pooling Layer* which further performs downsampling, reducing the number of parameters. Then they are connected to fully connected layers which works the same as ANNs.
CNNs are made up of neurons in 3 dimensions, height and width which is also called *Spatial dimension* and depth. They use special *Kernels* which are usually small in spatial dimension but deep in depth dimension. [2]

3 Results

In this section, we train different *Feed-Forward Neural Networks* which are trained on a low-resolution range of a function and have to predict the higher resolution original range.
The function we want to upscale is

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

First we will see the original graph which is the output of this function on 100 uniformly distributed points in the interval $[0, 1]$

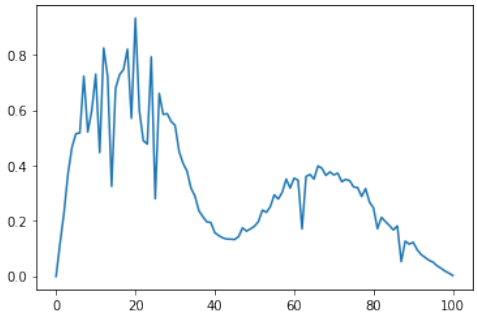


(a) Orignal output of $f(x)$ (b) $f(x)$ at every 20th point

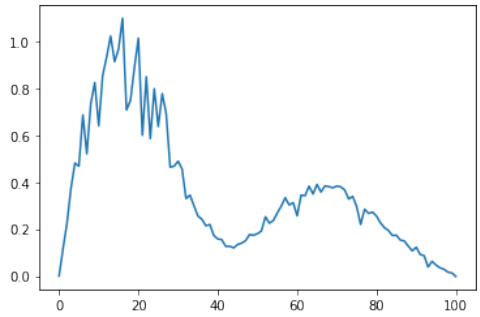
Figure 1. Different resolution datasets

Now we will start by training a basic *Feed Forward Neural Network* with this low-resolution data set with no hidden layer and also showing the comparison of adding a hidden layer.

For a fair comparison, we are training all the models with Tensorflow.keras.Sequential, for 100 epochs using MeanSquaredError loss function and keras Adam optimizer



(a) Output: 0 hidden layers

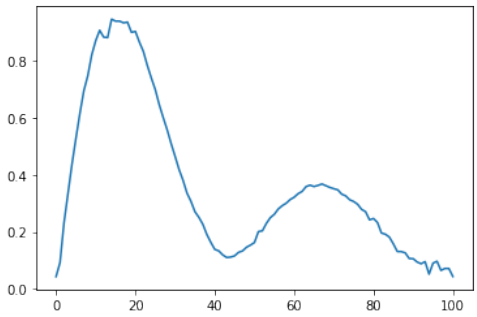


(b) Output: 1 hidden layer

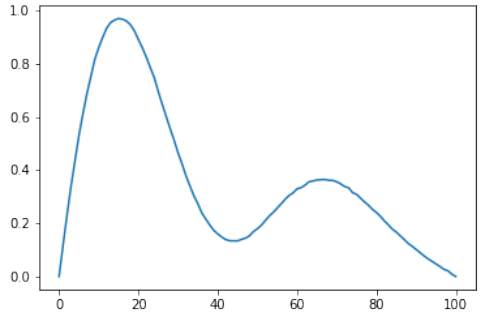
Figure 2. Upscaled from every 20th point

As we can observe from the figures, the distortion in the model with 1 hidden layer around the maxima is significantly lower than the model with 0 hidden layers.

Now we can see what happens when we train a model with similar architecture but with a comparitively higher resolution dataset.
Using every 10th and 5th point to train the next models.



(a) Upscaled from every 10th

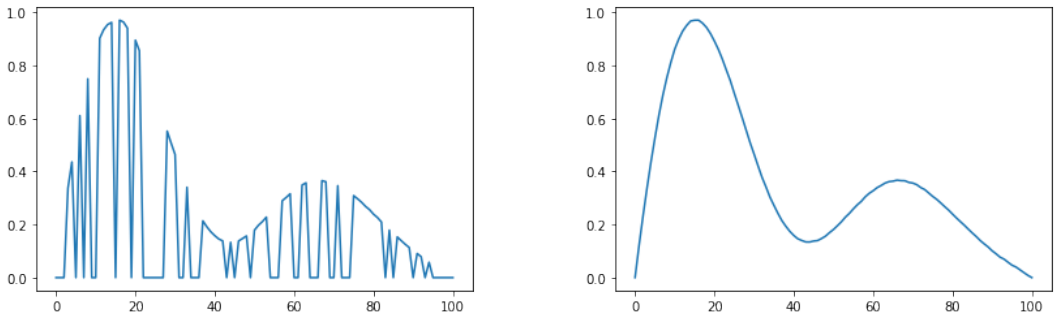


(b) Upscaled from every 5th point

Figure 3. Models using different training datasets

As we can observe from the results above, the quality of datasets is directly proportional to the quality of the predicted output.

Now to see the difference between activation functions for output layers.
We are going to use every 2nd point for the training dataset, which is almost similar to the original dataset. We might think that since it is so close to the original dataset, activation functions or other parameters will not make a significant difference in the final output.
Moving further, we can examine the difference in *ReLU* and *Linear* activation functions for the output layer.



(a) Model using ReLU

(b) Model using Linear

Figure 4. Models using different output layer Activation functions

As we can see, the Model using *ReLU* activation function for the output layer gives us output with significant error.

Why did that happen?

The inconsistency in the results is because *ReLU* activation function gives out 0 as output until the inner expression is positive, which is really helpful for classification-type deep learning models where the results are discrete. But in our use case, we are predicting a continuous function, which means a result with minimal error is more desirable for us rather than having a discrete result. Therefore, using *Linear or Sigmoid* activation function works best with our neural network.

We successfully upscaled our dataset with loss of 1.3332×10^{-6} for our best model consisting of 3 layers, 256 nodes in the hidden layer, using a combination of *ReLU* and *Linear* activation function, using *MeanSquaredError* and *Adam* optimizer.

4 Conclusion

We talked about *What is Deep Learning* and also discussed the basic structure of a neural network and briefed over some basic components of a simple neural network. Then we discussed different *Feed Forward Neural Networks* and worked around with a number of different designs for their creation and discussed the significance of choosing the dataset, number of hidden layers, choosing the activation function, loss function, and optimizer and compared different results when changing all those parameters.

Acknowledgements

Saurav Singh Chandel
The Author
biblio.bib

References

[1] LeCun Y., Bengio Y. and Hinton G., Deep learning, *nature* **521** (2015), 436–444.

[2] O'Shea K. and Nash R., An introduction to convolutional neural networks, *arXiv preprint arXiv:1511.08458* (2015).

[3] Sazli M.H., A brief review of feed-forward neural networks, *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering* **50** (2006).