Appendix: Automated assessment of residual plots with computer vision models

Weihao Li[a], Dianne Cook[a], Emi Tanaka[b,c], Susan VanderPlas[d], Klaus Ackermann[a]

[a]Department of Econometrics and Business Statistics, Monash University, Clayton, VIC, Australia; [b]Biological Data Science Institute, Australian National University, Acton, ACT, Australia; [c]Research School of Finance, Actuarial Studies and Statistics, Australian National University, Acton, ACT, Australia; [d]Department of Statistics, University of Nebraska, Lincoln, Nebraska, USA

ARTICLE HISTORY

Compiled May 28, 2024

1. Neural Network Layers Used in the Study

This study used seven types of neural network layers, all of which are standard components frequently found in modern deep learning models. These layers are well-documented in textbooks like Goodfellow, Bengio, and Courville (2016) and Chollet (2021), which offer thorough explanations and mathematical insights. In this section, we will offer a concise overview of these layers, drawing primarily from the insights provided in Goodfellow, Bengio, and Courville (2016).

1.1. *Dense Layer*

The Dense layer, also known as the fully-connected layer, is the fundamental unit in neural networks. It conducts a matrix multiplication operation between the input matrix $\boldsymbol{I}$ and a weight matrix $\boldsymbol{W}$ to generate the output matrix $\boldsymbol{O}$, which can be

CONTACT Weihao Li. Email: weihao.li@monash.edu, Dianne Cook. Email: dicook@monash.edu, Emi Tanaka. Email: emi.tanaka@anu.edu.au, Susan VanderPlas. Email: susan.vanderplas@unl.edu, Klaus Ackermann. Email: Klaus.Ackermann@monash.edu

written as

$$\boldsymbol{O} = \boldsymbol{I}\boldsymbol{W} + b,$$

where b is the intercept.

1.2. *ReLu Layer*

The ReLU layer, short for rectified linear unit, is an element-wise non-linear function introduced by Nair and Hinton (2010). It sets the output elements to zero if the corresponding input element is negative; otherwise, it retains the original input. Mathematically, it can be expressed as:

$$\boldsymbol{O}(i,j) = max\{0, \boldsymbol{I}(i,j)\},$$

where $\boldsymbol{O}(i,j)$ is the ith row and jth column entry of matrix $\boldsymbol{O}$, and $\boldsymbol{I}(i,j)$ is the ith row and jth column entry of matrix $\boldsymbol{I}$.

1.3. *Convolutaional Layer*

In Dense layers, matrix multiplication leads to each output unit interacting with every input unit, whereas convolutional layers operate differently with sparse interactions. An output unit in a convolutional layer is connected solely to a subset of input units, and the weight is shared across all input units. Achieving this involves using a kernel, typically a small square matrix, to conduct matrix multiplication across all input units. Precisely, this concept can be formulated as:

$$\boldsymbol{O}(i,j) = \sum_m \sum_n \boldsymbol{I}(i-m, j-n)K(m,n),$$

where m and n are the row and columns indices of the kernel K.

If there are multiple kernels used in one covolutional layer, then each kernel will have its own weights and the output will be a three-dimensional tensor, where the length of the third channel is the number of kernels.

1.4. *Pooling Layer*

A pooling layer substitutes the input at a specific location with a summary statistic derived from nearby input units. Typically, there are two types of pooling layers: max pooling and average pooling. Max pooling computes the maximum value within a rectangular neighborhood, while average pooling calculates their average. Pooling layers helps making the representation approximately invariant to minor translations of the input. The output matrix of a pooling layer is approximately s times smaller than the input matrix, where s represents the length of the rectangular area. A max pooling layer can be formulated as:

$$\boldsymbol{O}(i,j) = \max_{m,n} \boldsymbol{I}(si+m, sj+n).$$

1.5. *Global Pooling Layer*

A global pooling layer condenses an input matrix into a scalar value by either extracting the maximum or computing the average across all elements. This layer acts as a crucial link between the convolutional structure and the subsequent dense layers in a neural network architecture. When convolutional layers uses multiple kernels, the output becomes a three-dimensional tensor with numerous channels. In this scenario, the global pooling layer treats each channel individually, much like distinct features in a conventional classifier. This approach facilitates the extraction of essential features from complex data representations, enhancing the network's ability to learn meaningful patterns. A global max pooling layer can be formulated as

$$O(i,j) = \max_{m,n,k} I(si+m, sj+n, k),$$

where k is the kernel index.

1.6. *Batch Normalization Layer*

Batch normalization is a method of adaptive reparametrization. One of the issues it adjusts is the simultaneous update of parameters in different layers, especially for network with a large number layers. At training time, the batch normalization layer normalizes the input matrix I using the formula

$$\boldsymbol{O} = \frac{\boldsymbol{I} - \boldsymbol{\mu}_I}{\boldsymbol{\sigma}_I},$$

where $\boldsymbol{\mu}_I$ and $\boldsymbol{\sigma}_I$ are the mean and the standard deviation of each unit respectively.

It reparametrizes the model to make some units always be standardized by definition, such that the model training is stabilized. At inference time, $\boldsymbol{\mu}_I$ and $\boldsymbol{\sigma}_I$ are usually replaced with the running mean and running average obtained during training.

1.7. *Dropout Layer*

Dropout is a computationally inexpensive way to apply regularization on neural network. For each input unit, it randomly sets to be zero during training, effectively training a large number of subnetworks simultaneously, but these subnetworks share weights and each will only be trained for a single steps in a large network. It is essentially a different implementation of the bagging algorithm. Mathematically, it is formulated as

$$\boldsymbol{O}(i,j) = \boldsymbol{D}(i,j)\boldsymbol{I}(i,j),$$

where $\boldsymbol{D}(i,j) \sim B(1,p)$ and p is a hyperparameter that can be tuned.

References

Chollet, Francois. 2021. *Deep learning with Python*. Simon and Schuster.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

Nair, Vinod, and Geoffrey E Hinton. 2010. "Rectified linear units improve restricted boltzmann machines." In *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807–814.