Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea Magistrale in Informatica

Construction of a robust pipeline to estimate 3D trajectories from 2D hand landmarks

Relatore: Dr. Alessandro Giusti

Co-relatore: Dr. Loris Roveda

Co-relatore: Dr. Gianluigi Ciocca

Tesi di Laurea Magistrale di:
Umberto Cocca
Matricola 807191

Anno Accademico 2020-2021

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Umberto Cocca
Lugano, 23 February 2022

-dedica-

“Gutta cavat lapidem.”
lat. «la goccia scava la pietra»

Abstract

Robotics systems are increasingly adopted for filming and photography purposes. Exploiting robots, in fact, it is possible to program complex motions for the camera, achieving high-quality video/photography.

These robots, until recent years, must be controlled through physical remote systems, nevertheless, this is not true anymore. Thanks to the latest efforts, a high-fidelity hand and finger tracking solution without specialized hardware needed (e.g. depth sensors, Kinect etc...) was built by MediaPipe machine learning solutions [Zhang et al., 2020].

Delving into the goal of capturing 3D motions taking advantage of a hand tracking system, my contribution was, in the first instance, to build a hand gestures recognition neural network, next to set up a robust pipeline that solves different problems during the process of detecting 3D trajectories obtained from 2D pixel landmarks as estimating orientation (roll, yaw and pitch) and consequently z-space as a result of orientation test algorithm and matrices transformations. The trajectories are based on a dynamic speed, interpolating and smoothing them with ridge regression.

In conclusion, the captured trajectory has been launched on a drone in simulation. It is implemented in the ROS framework, using Gazebo as a tool by reason of robust physics engine with high-quality graphics.

Since hand tracking is a vital component to provide a natural way for interaction and communication in AR/VR then the entire pipeline that has been built to detect the trajectories can be easily translated into another kind of task.

Acknowledgements

Work in progress

Contents

Figures

Tables

Equations

Listings

Introduction

Chapter 1

Literature review

This chapter discusses previous research about the topic, providing a brief introduction to the approach we adopted.

Chapter 2

Background

This chapter provides some background concepts to understand the material presented in this thesis.

2.1 Regression

The goal of regression is to predict the value of one or more continuous target variables t given the value of a D-dimensional vector x of input variables. [Bishop, 2006]
Given a training data set comprising N observations x_n, where $n = 1, ..., N$, together with corresponding target values t_n, the goal is to predict the value of t for a new value of x.
From a probabilistic perspective, we aim to model the predictive distribution $p(t|x)$ because this expresses our uncertainty about the value of t for each value of x.

2.1.1 Linear Models for Regression

The simplest linear model for regression is one that involved a linear combination of the input variables:

$$y(x, w) = w_0 + w_1 x_1 + ...w_D x_D \tag{2.1}$$

where $x = (x_1, ..., x_D)^T$. This is often simply known as linear regression. The key property of this model is that it is a linear function of the parameters w_i, but also of x_i and establishes significant limitations on the model. It is possibile extend the class of models by considering linear combinatioins of fixed nonlinear functions of the input variables:

$$y(x,w) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(x) \tag{2.2}$$

where $\phi_j(x)$ are known as basis functions. The total number of parameters in this model will be M.
The parameter w_0 is called bias parameter. It is often convenient to define an additional dummy "basis function" $\phi_0(x) = 1$, so that:

$$y(x,w) = \sum_{j=0}^{M-1} w_j \phi_j(x) = w^T \phi(x) \tag{2.3}$$

where $w = (w_0, ..., w_{M-1})$ and $\phi = (\phi_0, ..., \phi_{M-1})^T$
By using non linear basis functions, we allow the function y(x,w) to be a non linear function of the input vector x. A particle example of this model where there is a single input variable x is the polynomial regression. The basis functions take the form of powers of x so that $\phi_j(x) = x^j$.
There are other possibile choices for the basis functions as:

$$\phi_j(x) = e^{\frac{-(x-u_j)^2}{2s^2}} \tag{2.4}$$

where u_j regulates the locations of the basis functions in input space, while the parameter s is their spatial scale. These are usally reffered to as "Gaussian" basis functions.
We can use simply the identity basis functions in which the vector $\phi(x) = x$.

2.1.2 Normal equations

The values of the coefficients will be determined by fitting the polynomial to the training data. This can be done by minimizing an error function that measures the misfit between the function $y(x, w)$, for any given value of w, and the training set data points. One simple choice of error function, which is widely used, is given by the sum of the squares of the errors between the predictions $y(x_n, w)$ for each data point x_n and the corresponding target values t_n (called also sum of squares regression [Valchanov, 2018]), so that we minimize:

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}[y(x_n, w) - t_n]^2 \tag{2.5}$$

where the factor of 1/2 is included for mathematical convenience. It is a nonnegative quantity that would be zero if, and only if, the function $y(x, w)$ were to pass exactly through each training data point.

We can solve the curve fitting problem by choosing the value of w for which $E(w)$ is as small as possible. Because the error function is a quadratic function of the coefficients w, its derivatives with respect to the coefficients will be linear in the elements of w, and so the minimization of the error function has a unique solution w^*. The resulting polynomial is given by the function $y(x, w^*)$.

We can write (2.5) in matrix notation as:

$$\frac{1}{2}(\phi w - t)^T(\phi w - t) \tag{2.6}$$

where ϕ is an NxM matrix, so that:

$$\phi = \begin{pmatrix} \phi_0(x_1) & \phi_1(x_1) & \dots & \phi_{M-1}(x_1) \\ \phi_0(x_2) & \phi_1(x_1) & \dots & \phi_{M-1}(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \dots & \phi_{M-1}(x_N) \end{pmatrix} \tag{2.7}$$

Equation 2.1. This is called the design matrix whose elements are given by $\phi_{nj} = \phi_j(x_n)$.

$$w = \begin{pmatrix} w_0 \\ \vdots \\ w_{M-1} \end{pmatrix} \quad t = \begin{pmatrix} t_1 \\ \vdots \\ t_N \end{pmatrix} \tag{2.8}$$

Therefore, we have to solve the optimization problem finding the minimum of the cost function $E(w)$:

$$w^* = \arg\min_w \frac{1}{2}(\phi w - t)^T(\phi w - t) \tag{2.9}$$

So we compute the gradient and solving for w we obtain:

$$\begin{aligned} \nabla E(w) = \phi^T(\phi w - t) = 0 \\ \phi^T\phi w - \phi^T = 0 \\ \phi^T\phi w = \phi^T t \\ w^* = (\phi^T\phi)^{-1}\phi^T t \end{aligned} \tag{2.10}$$

Equation 2.2. They are known as the normal equations for the least squares problem

This is the real minimum because if we take the second derivative $\nabla^2 E(w) = \phi^T\phi$ and this is a symmetric matrix, so it's also positive definitive matrix, which means that this objective function that we try to minimize is convex. So if we find a stationary point, such that derivative is zero, we also find a global minium. Furthermore, to find a solution we need to invert this matrix $\phi^T\phi$, so we need some condition that assure this is invertible and this is the case when the columns of the matrix are linearly independent.
Once we find the solution w^* and we receive a new data point that we never seen during training, we just predict the new target t^* as:

$$t^* = {w^*}^T \phi(x) \tag{2.11}$$

2.2 Regularization

If we use a set of features that is too expressive, for example a ten polynomial grade (x^{10}), then this interpolates very close our training data, because we have a model with 10 parameters where we can perfectly represents the data points. The risk is that the model became something like this:
[picture]
We want a trade-off between fits data and able to generalize. Infact, on the other hand if our model is not to expressive and not to complex our data will be linearly

rapresentable in the feature space and this means that the performance will be very poor.

2.2.1 The Problem Of Overfitting

We want to penalize for some features with parameters with high values, if our model is overfitting is very likely that the parameters of our model will have a big magnitude, this means that also the features supply to this parameters will be higher and very low and this cause a lot of problems.
In order to control over-fitting we introduce the idea of adding a regularization term to an error fuction, so that the total error function to minimized takes the form:

$$E(w) = E_D(w) + \lambda E_W(w) \tag{2.12}$$

Where λ is the regularization coefficient and it is the trade-off between how we well fit training set and how to establish the parameters w with low values, therefore having simple hypotesys avoiding over-fitting. E_D is the error based on dataset, while E_W is based on weights.

2.2.2 Cost Function

One of the simplest forms of regularizer is given by the sum-of-squares of the weight vector elements:

$$E_W(w) = \frac{1}{2}w^T w = \frac{||w||_2^2}{2} \tag{2.13}$$

where $||w||_2$ is the euclidean norm $\sqrt{\sum_{i=1}^{n} x_i^2}$.
This is also called ridge regression, a method of estimating the coefficitents of multiple-regression models in scenarios where indipendent variables are highly correlated.
If we also consider the sum-of-squares error function given by:

$$E_D(w) = \frac{1}{2}[t_n - w^T \phi(x_n)]^2 \tag{2.14}$$

then the total error function becomes:

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}[t_n - w^T\phi(x_n)]^2 + \frac{\lambda}{2}w^T w \tag{2.15}$$

This particular choice of regularizer is known in the machine learning literature as weight decay because in sequential learning algorithms, it encourages weight values to decay towards zero.

Setting the gradient of $E(w)$ wrt w to zero, and solving for w, we obtain:

$$w^* = (\lambda I + \phi^T\phi)^{-1}\phi^T t \tag{2.16}$$

and this is an extension of the least-squares solution (2.10). This is a better version than before because is also possibile prove that $(\lambda I + \phi^T\phi)$ is always invertible if $\lambda > 0$, therefore w^* always exists.

2.2.3 (Batch) Gradient Descent

2.3 Artificial Neural Networks

The term Neural Network (NN) has its origins in attempts to find mathematical representations of information processing in biological systems. In fact, Artificial Neural Networks (ANNs), subset of Machine Learning (ML) field, are models inspired from the biological performance of human brain [Andina et al., 2007].

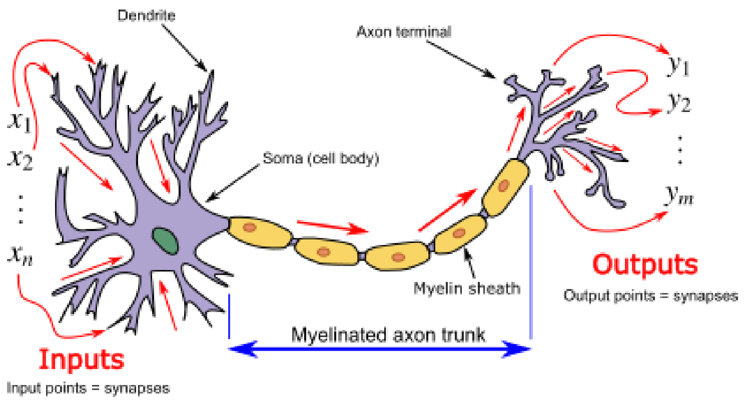

Figure 2.1. Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals. The signal represents a short electrical pulse called 'spike'.

Neurons have cel body (fig. 2.1) and a number of input wires called dendrites. Neurons also have an output wire called axon, used to send signals to other neurons. At a simplistic level the neuron is a computational unit that gets a number of inputs through its input wires, then does some computation and finally it sends outputs to other neurons connected to it in the brain.

2.3.1 Feedforward Fully-Connected Neural Networks

In (fig. 2.2) is visible a NN, seen as mathematical model. It's just a group of this different neurons strung together. Trying to underline an analogy with the biological systems: circles identify the cell body where they are feeded with some inputs that pass through the input wires, similar to the dendrites. The neuron does some computation and outputs some value on an output wire, where in the biological neuron it identifies the axon.

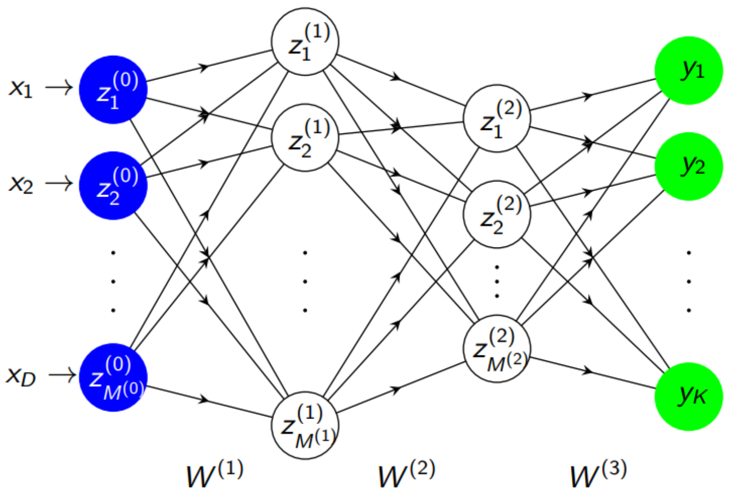

Figure 2.2. The image graphically shows a NN with three layers (or two hidden layers). The input layer has $M^{(0)}$ neurons and the output layer has K outputs. Neurons are organized in layers to allow parallel copmutation to avoid cyclic dependendices. The process of computing NN outputs from inputs is called forward propagation. This kind of network is also called Multi-layer perceptron.

The $z_m^{(l)}$ are neurons, each takes its input values and computes a single output value from them. Neurons are organized in layers $1, \ldots, L$ and usually the starting input is considered the 0th layer. Inputs $x_1, \ldots, x_D$ are occasionally called input layer/neurons (even though they do not compute anything). The output of the entire network is then $y = z^{(L)}$, called output layer. Instead, the internal layers are called hidden layers. Each layer $l \in \{1, \ldots, L\}$ has $M^{(l)}$ neurons.
$M^{(1)}$ neurons perform a preceptron-like computation:

$$u_m^{(1)} = (w_m^{(1)})^T x + b_m^{(1)}, \qquad z_m^{(1)} = f(u(1)_m), \qquad m = 1, \ldots, M^{(1)} \tag{2.17}$$

with a differentiable activation function f for gradient descent. This step is iterated multiple times taking the outputs of the previous step:

$$z^{(l-1)} = (z_m^{(l-1)})_{m=1,\ldots,M^{(l-1)}} \tag{2.18}$$

as input of:

$$u_m^{(l)} = (w_m^{(l)})^T z^{(l-1)} + b_m^{(l)}, \qquad z_m^{(l)} = f(u_m^{(l)}) \tag{2.19}$$

where $m = 1, ..., M^{(1)}$ and $l = 2, ..., L$. Weights w are usually indendent for each step. Additionally define $z^{(0)}$ to be the input, i.e.

$$z^{(0)} = x \tag{2.20}$$

For each layer $l \in 1, ..., L$ the computation is

$$z_m^{(l)} = f((w_m^{(l)})^T z^{(l-1)} + b_m^{(l)}) \tag{2.21}$$

which can be written as a matrix multiplication:

$$z^{(l)} = f(W^{(l)} z^{(l-1)} + b^{(l)}) \tag{2.22}$$

Equation 2.3. Function that idetifies input transformation at each step l of the net.

The weights w are directed connections between the neurons, e.g. the neurons of layer 2 are connected to the ones of layer 1 by the weights $w_{mn}^{(2)}, m = 1, ...M^{(1)}, n = 1, ..., M^{(2)}$. The bias b varies according to the propensity of the neuron to activate, influencing its output.
$f()$ is an activation functions and need to be differentiable, because we wish to apply gradient descent training. For the hidden layers of the nework, the activation function must be nonlinear, because multiple linear computations can be collapsed to a single one, therefore in order to gain power from iterative computation we thus need nonlinear steps.
Many possible activation functions for the hidden layers of a NN exist:

- Sigmoid, Hyperbolic Tangent: Monotonic, squeeze output to a fixed range
- ReLU: "almost linear" (a clipped identity function)

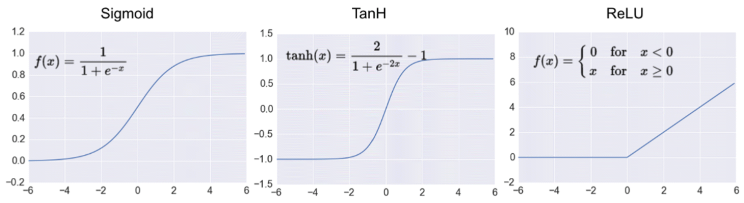

Figure 2.3. Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals. The signal represents a short electrical pulse called 'spike'.

One of the key characteristics of modern deep learning system is to use non-saturated activation function (e.g. ReLU) to replace its saturated counterpart (e.g. sigmoid, tanh). The advantage of using non-saturated activation function lies in two aspects: the first is to solve the so called "exploding/vanishing gradient" [Bengio et al., 1994], in particular on the difficulty of training Recurrent Neural Network (RNN) [Pascanu et al., 2013], while the second is to accelerate the convergence speed. More sophisticated activation function as the "leaky ReLU" trying to solve the dying ReLU problem [Lea]. In contrast to ReLU, in which the negative part is totally dropped, leaky ReLU assigns a noon-zero slope to it. Leaky ReLU and its variants are consistently better than ReLU in Convolutional Neural Network (CNN) [Xu et al., 2015].

Leaky Rectified linear activation is introduced in acoustic model [Maas et al.]. Mathematically, it is defined as follows:

$$f(x) = \begin{cases} \frac{x}{a} & \text{if } x < 0 \\ x & \text{otherwise} \end{cases} \tag{2.23}$$

Equation 2.4. Function that idetifies input transformation at each step l of the net.

where a is a fixed parameter in range $(1; +\text{inf})$. In original paper, the authors suggest to set a to a large number like 100.

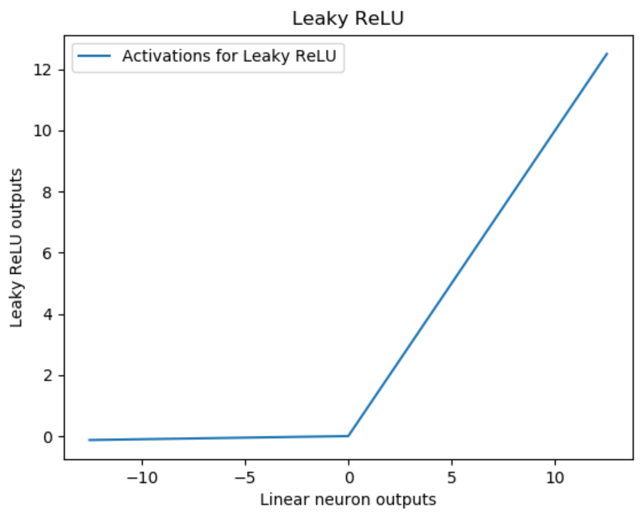

Figure 2.4. Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals. The signal represents a short electrical pulse called 'spike'.

2.3.2 NN Setup for Classification

For a classification task with K classes, we use a K-idmendional output layer. A sample $x \in R^D$ is classified as belonging to class k if the output neuron y_k has the maximal value:

$$c^* = \arg\max_k y_k \tag{2.24}$$

The problem is that the $argmax$ function is not differentiable. Therefore, this is solved by letting the NN output a probability distribution over classes:

$$y = (y_k)_{k=1,\dots,K} \qquad y_k \geq 0, \quad \sum_k y_k = 1 \tag{2.25}$$

Equation 2.5. Function that idetifies input transformation at each step l of the net.

The advantage is that we can derive a differentiable measure of the quality of the output on theoretical grounds, using probability theory. In order to make the network output a probability distribution, we take exponentials and normalize. This is the softmax nonlinearity:

$$S(y) = (\frac{e^{y_1}}{\sum_k e^{y_k}}, ..., \frac{e^{y_K}}{\sum_k e^{y_k}}) \tag{2.26}$$

Equation 2.6. Function that idetifies input transformation at each step l of the net.

In contrast to other activation functinos, it is applied to the full last layer of the NN, not to each indipendent component. The hidden layers can have any nonlinear activation function.

The learning process is structured as a non-convex optimisation problem in which the aim is to minimise a cost function, which measures the distance between a particular solution and an optimal one.

If we assume a NN with softmax output, we can compute the loss by measuring the cross-entropy between the output distribution and the target distribution.

Encoding the target in one-hot style, e.g. if a sample belongs to class k, the target is:

$$t = (0, ..., 0, 1, 0, ..., 0) \tag{2.27}$$

Equation 2.7. Function that idetifies input transformation at each step l of the net.

We treat this as a probability distribution: in an ideal world, a perfect hypothesis y would exactly match this t, assigning probability 1 to the correct class, and probability 0 otherwise. The cross-entropy loss is defined as:

$$E_{CE} = -\sum_k (t_k \log y_k) \tag{2.28}$$

Equation 2.8. Function that idetifies input transformation at each step l of the net.

The intuition about the cross-entropy corresponds to the number of additional bits needed to encode the correct output, given that we have access to the (possibly wrong) prediction of the network. One property of the cross-entropy loss is that is always non-negative. For efficiency and numerical stability, one should merge softmax loss and cross-entropy criterion into one function:

$$E_{CE+SM} = -\sum_k (t_k \log S_k(y)) \qquad (2.29)$$

Equation 2.9. To train the network with backpropagation, you need to calculate the derivative of the loss. In the general case, that derivative can get complicated, but using the softmax and the cross entropy loss, that complexity fades away.

2.3.3 Optimisation algorithms

The choice of optimization algorithms strongly influence the effectiveness of the learning process as they update and calculate the appropriate and optimal values of that model. Specifically if we consider the gradient descent, the most popular optimization strategy used in machine learning, the extent of the update is determined by the learning rate λ, which guarantees convergence to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces. By the way, there are better optimization as the non linear conjugate gradient [Hager and Zhang, 2006], BFGS, the improved version to decrease memory usage L-BFGS [Saputro and Widyaningsih, 2017], etc... the main advantages are that no need to manually pick λ and often are faster than gradient descent, although more complex algorithms.

The optimiser chosen for this thesis project is Adam, an algorithm for first-order gradient-based optimisation of stochastic objective functions, based on adaptive estimates of lower-order moments [Kingma and Ba, 2017]. It is an extension to stochastic gradient descent that has recently seen broader adoption for Deep Learning (DL) applications in computer vision and Natural Language Processing (NLP).

Chapter 3

Tools

Chapter 4

Methodologies

Chapter 5

Evaluation

Conclusion and perspectives

Appendix A

List of Acronyms

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DL	Deep Learning
ML	Machine Learning
NLP	Natural Language Processing
NN	Neural Network
RNN	Recurrent Neural Network

Bibliography

Leaky relu: improving traditional relu – machinecurve. `https://www.machinecurve.com/index.php/2019/10/15/leaky-relu-improving-traditional-relu/`. (Accessed on 01/11/2022).

Diego Andina, D. Pham, D. Andina, Antonio Vega-Corona, Juan Seijas, and J. Torres-Garcìa. *Neural Networks Historical Review*. 02 2007. ISBN 978-0-387-37450-5. doi: 10.1007/0-387-37452-3_2.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

W. W. Hager and H. Zhang. A survey of nonlinear conjugate gradient methods, 2006. Pacific journal of Optimization, vol. 2, no. 1, pp. 35–58.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. Citeseer.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.

Dewi Retno Sari Saputro and Purnami Widyaningsih. Limited memory broyden-fletcher-goldfarb-shanno (l-bfgs) method for the parameter estimation on geographically weighted ordinal logistic regression model (gwolr). In *AIP Conference Proceedings*, volume 1868, page 040009. AIP Publishing LLC, 2017.

Iliya Valchanov. Sum of squares total, sum of squares regression and sum of squares error, 2018. URL `https://365datascience.com/tutorials/statistics-tutorials/sum-squares/`. [sum of squares total, denoted SST | Published on 5 Nov 2018].

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking, 2020.