Machine Learning Techniques For The Estimation of The Operating Parameters of Solar Cells

Papadomichelakis Georgios
University of Crete
Department of Mathematics and Applied Mathematics
temp52@math.uoc.gr

Abstract

We consider the problem of predicting the internal temperature in photovoltaic cells depending on ambient and/or internal factors. In this thesis we use machine learning techniques, specifically deep learning and neural networks to accurately forecast the temperature using methods we developed. We present an introduction to the mathematical background of neural networks and build some using the Python3 programming language and TensorFlow. Lastly we present the numerical results comparing what our neural networks managed to predict to the actual temperatures measured.

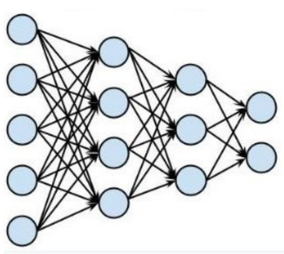
Keywords:

- Machine Learning
- Deep Learning
- Artificial Neural Networks
- Neurons
- Weights & Biases
- Activation Functions
- Optimization
- Gradient Descen

Introduction

During the recent years, there is an ever-growing research interest on *Artificial Intelligence* (AI). Another term we hear about, more and more often, is *Machine Learning* (ML) which can be seen as a subset to AI. *Neural Networks* (NNs) and *Deep Learning* (DL) are both included in that subset of Machine Learning. This thesis is about using *Artificial Neural Networks* (ANNs) and deep learning methods, in order to deal with a problem coming from the photovoltaic cells' industry. In particular we are interested in developing supervised machine learning techniques for estimating the operating temperature of photovoltaic cells (PV cells) using various parameters.

Artificial Neural Networks



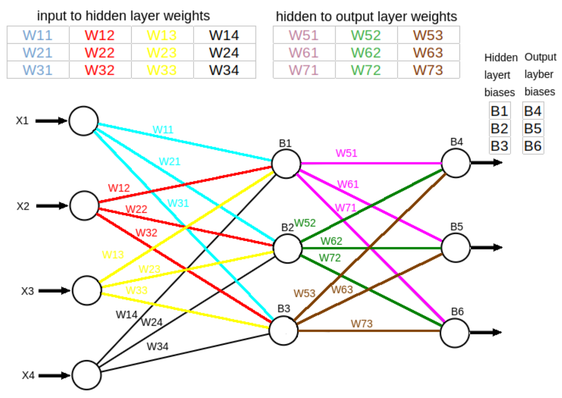
A *feedforward* neural network is an ANN which is described by an algorithm working in layers where the connections between the neurons do not form a cycle and the data are passing from the input neurons to the adjoined neurons and so on
In order to define a neuron we need a set of synapses, each characterized by a weight. Specifically, a piece of data x_j at the input of synapse j, connected to the neuron k, is multiplied by the synaptic weight w_{kj}.
We can describe the neuron k of an ANN as following:

$$z_k = \sum_{j=1}^{m} w_{kj}x_j + b_k,$$

where w_{kj} is the corresponding weight for the neuron k and synapse j and b_k is the bias term.
After the summation is completed a mathematical function, called activation function is applied. The output of that neuron, after the data have been processed and the activation functions have fired, are set as

$$y_k = a(z_k),$$

where $a(\cdot)$ is the activation function.
An ANN is described by one input layer, one output layer which gives the final predictions of the model and one or more hidden layers between the input and output layers.
We represent all the synaptic weights connecting all neurons of a given layer fully connected to its previous layer, by a matrix of weights. The notation for the biases is simpler and can be expressed as a vector, $b^n \in R^m$.



The above figure is a visualization of the notation we used so far.

Universal Approximation Theorem
Let σ be any continuous sigmoidal function. Given any function $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum

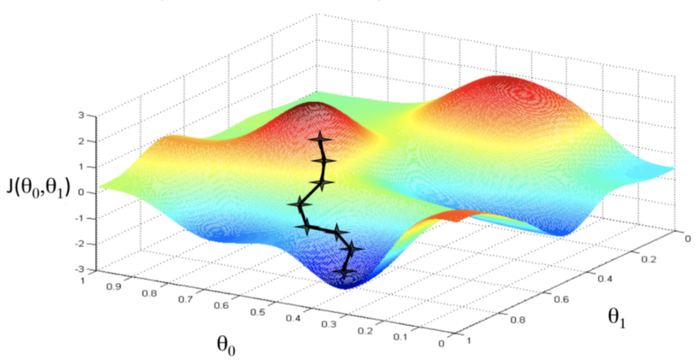
$$G(x) = \sum_{j=1}^{N} c_j \sigma(w_j^T x + b_j)$$

for which

$$|G(x) - f(x)| < \varepsilon \quad \forall x \in I_n$$

In other words, the finite sums of the above form are dense in $C(I_n)$ ■

The goal of any algorithm in deep learning is to minimize a cost function. A cost function is a mechanism that returns the error between predicted outcomes compared with the actual outcomes. We can view this cost function as a mountain where high altitude means big errors and low altitude means small errors.



Gradient Descent

In order to minimize the cost, machine learning algorithms most often use the Gradient Descent algorithm. Gradient descent, indicates where our next step should head for. We could summarize the process in the following steps:

- Start with a random point/vector p of parameters θ_i.
- Repeatedly calculate the gradient $\nabla J(p)$, take small steps in that direction and update the parameters using $p \coloneqq p - \alpha \nabla J(p)$ until we (hopefully) converge to a minimum. The parameter α is called the *learning rate* i.e. the length of the step.

Implementation/Methodology

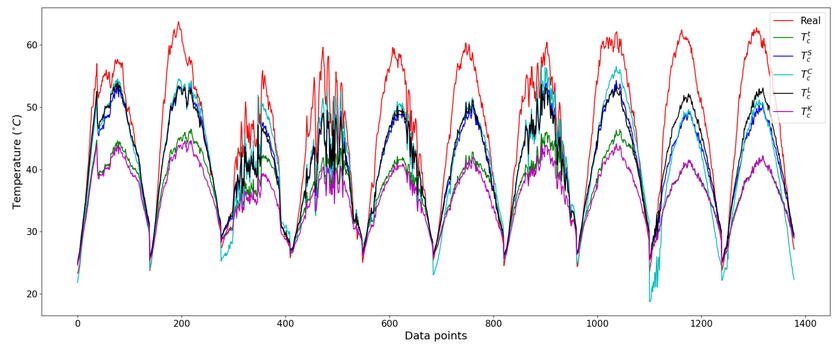
We implement three different methods to approximate the temperature of the cells, differing only in the data used as inputs of the ANNs.

- First Method Data Used:
  - solar irradiance $G_{irr}(w/m^2)$
  - air temperature $T_{air}(C)$
  - wind speed $W(m/s)$
  - short circuit current $I_{sc}(A)$
  - open circuit voltage $V_{oc}(V)$
- Second Method Data Used:
  - solar irradiance $G_{irr}(w/m^2)$
  - air temperature $T_{air}(C)$
  - wind speed $W(m/s)$
  - Normal Operating Cell Temperature $T_{NOCT}(C)$
  - module's efficiency $n_{ref}(14.5\%)$
- Third Method Data Used:
  - Results from 5 different empirical formulas for estimating the cell's operating temperature

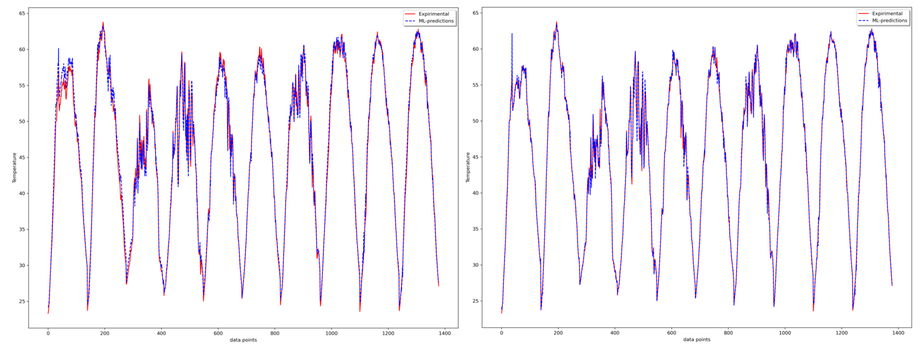
Empirical Formulas

$$T_c^t = T_{air} + \frac{G_{irr}}{800}(T_{NOCT} - 20)(1 - n_{ref})\left(\frac{9.5}{5.7 + 3.8W}\right)$$
$$T_c^S = T_{air} + 0.0138G_{irr}(1 + 0.031T_{air})(1 - 0.042W)(1 - 1.053n_{ref})$$
$$T_c^C = 0.943T_{air} + 0.028G_{irr} - 1.528W + 4.3$$
$$T_c^L = 30.006 + 0.0175(G_{irr} - 300) + 1.14(T_{air} - 25)$$
$$T_c^K = T_{air} + G_{irr}e^{-3.473-0.0594W}$$

These formulas are extremely inaccurate so we need to find a better combination of those. We can see how inaccurate they are in the figure below.



Numerical Results



In the figure above we can see the graphs of two different DNNs, comparing the observed temperature (red) with the network's predictions (blue) for the first method. On the left side of the figure we have the predictions of the network with 2 hidden layers and 5 neurons in each hidden layer, while on the right side we have the predictions of the network with 5 hidden layers and 40 neurons in each hidden layer.

HL / Nr	2	3	4	5
5	0.711	0.623	0.602	0.609
10	0.589	0.578	0.532	0.447
15	0.630	0.463	0.435	0.375
20	0.619	0.629	0.392	0.396
30	0.684	0.445	0.326	0.299
40	0.567	0.530	0.322	0.229

HL / Nr	2	3	4	5
5	1.233	1.182	1.074	1.109
10	1.106	0.863	0.945	0.806
15	1.067	0.771	0.663	0.699
20	0.928	0.659	0.564	0.471
30	0.978	0.609	0.563	0.622
40	0.963	0.604	0.475	0.424

HL / Nr	2	3	4	5
5	1.351	1.396	1.210	1.316
10	1.426	1.085	1.005	1.063
15	1.276	0.958	0.860	0.797
20	1.316	0.937	0.747	0.677
30	1.262	0.866	0.644	0.654
40	1.275	0.786	0.654	0.430

The tables above, depict the mean absolute errors for each method used (1,2,3) starting from left to right, for various numbers of neurons (Nr) and various numbers of hidden layers (HL)