Estimation of solar energy is a task with many benefits to a diverse group of people. This purpose is pursued with many different methodologies. Artificial Neural Networks (ANNs) are the novel methods of choice in the last decade. We compare the classical solar irradiation estimation methods with different ANN schemes including different inputs, data amount and estimation target. Our analyses show that the use of ANN to predict solar irradiation reaching the Earth’s surface gives similar results with that of the classical regression approaches. The small difference between these two approaches lies within the instrumentation accuracy of the measuring devices.

Machine Learning (ML) is a cluster of algorithms that are used to obtain models from data. Current state of the art consists of artificial neural networks (or just neural networks) that can learn useful representations from data or environment interaction. Machine learning is not a new methodology, having gone through a boom and bust cycle[9]. Current resurgence is attributed to more data being available, more efficient algorithms and hardware improvements. Machine learning consists of various different approaches that can be quite distinct from each other. Constructing a walker with a few sticks and joints then evolving them according to a fitness criteria (e.g distance covered) is under the umbrella of machine learning, as well as video recommendation engines and spam detectors. There are two main components to a machine learning application. First is the model and second is the optimization procedure to optimize the model by changing the parameters based on data. Our model is a neural network (specifically a multi layer perceptron) while our optimization procedure is a variant of gradient descent called AdaGrad[10]. Our type of problem is called a regression problem as opposed to a classification problem. In instances where we’re aiming to approximate a continuous function it’s called regression, an example of which is estimation of housing prices. If discrete labels or categories are output then it is termed classification which would be exemplified by email spam filters. 9 There are also two different kinds of learning. One is where “true” results are available (as is in our case or spam filters) which is called “supervised learning” and where such data is not available (like video recommendation) which is called “unsupervised learning”. As for the optimization procedure, backpropagation and variants of gradient descent are the current staples of training a neural network. Gradient descent can be understood via an analogy to gravity. Over a surface (called loss surface) a point particle is dropped under gravity. As such the particle will move in the direction of lower elevation, eventually settling to a (often local) minima if there is one. The surface is an analogy to how similar obtained model is to what the data represents, whereas gradient descent is analogous to gravity. On the surface higher points are where the model and data diverge, lower points are where they agree. The main task of machine learning is to find the lowest possible valley. Of course the dimensionality of neural networks is significantly greater than 3, but the analogy still remains valid. To wit, this work is about supervised training of an artificial neural network to obtain a model to estimate daily solar irradiation with different input variables using the aforementioned AdaGrad as the optimization algorithm. 3.2 Neural Networks Neural networks are originally inspired from the structure of our brains, with neurons connecting to each other and sending signals if neuron is “active”. Software equivalent of this structure is expressed through a “graph” (as in graph theory, mathematics of pairwise connected objects). An example graph is provided in figure 3.1. In figure 3.2 you can see an example of a neural network with 2 hidden layers. Each circle is a “node” corresponding to a neuron. Connections are made between one group of neurons (layer 1 neurons) to another group (layer 2 neurons). This structure of layered units, with inputs connecting to inner neurons which in turn are connected to the output node is the most basic structure of a neural network with a single hidden layer. These inner neurons are called “hidden” because they don’t interact with input or output data directly. Any neural network with more than 1 hidden layer is called a 10 Figure 3.1: © Taiyaki1228 (https://commons.wikimedia.org/wiki/File:Wikibooks\_graph\_theory.png), Wikibooks graph theory, https://creativecommons.org/licenses/by-sa/3.0/legalcode An example graph, with 5 nodes labeled with vi . In a mathematical contexts the name given to nodes is vertex, the connections are called edges. In machine learning we will call them neurons and weights, respectively. Also, we will not have connections like the one between v1 and v3 in a neural network since a connection like that prohibits formation of layered neurons. 11 Figure 3.2: © John Salatas (https://commons.wikimedia.org/wiki/File:Multilayer\_Neural\_Network.png), Multilayer Neural Network, https://creativecommons.org/licenses/by-sa/3.0/legalcode A neural network consisting of 2 hidden layers. It has 3 inputs and 2 outputs. Unlike the general graph, this structure has clear layers which can be deduced from the lack of connections between neurons in the same layer. deep neural network. Similar to a neuron activating, our mathematical model needs a gating function to characterize activity of a node. Most historically appropriate function for this purpose is called the sigmoid function while the most recent and state of the art choice is rectified linear unit (relu). Another option for a neuron’s activation is hyperbolic tangent. These are shown in fig 3.3. The capacity of neural networks to approximate any continuous function is established via the universal approximation theorem. Chapter 4 of Michael Nielsen’s book [11] demonstrates a graphical approach to how neural networks can approximate any continuous function. There are also analytical proofs from Cybenko [12] and Hornik et al. [13]. As an example, for a single layer neural network with 5 inputs, 10 hidden neurons and 1 output the entire model under consideration is as follows: 12 Figure 3.3: Activation of a single neuron as a function of its input. 13 N = XW2iai + b2 (3.1a) ai = σ( XW1jXj + b1) (3.1b) where W1 and W2 are connection weights, and b1, b2 are the bias terms. W1 is for the connections from input to hidden layer and W2 for the connections from hidden layer to output. As we have 5 inputs and 10 hidden neurons, shape of W1 is (5, 10) as a matrix. Shape of W2 is (10, 1) so in total this network contains 62 parameters, including two bias terms. Neural networks are initialized with random weights, so their output is completely random in the first run. The only meaning of the formulation comes through universal approximation theorem. Once a neural network learns from data, the resulting model is encoded in the activity of the nodes. This means neural networks are black boxes by default. Their universality, however, makes them worthwhile. 3.3 Gradient Descent Once we have a neural network with randomly initialized weights, we are supposed to optimize the weights to obtain a model that matches given data. Optimization is a process aimed at a goal. In the case of machine learning that goal is called loss function, or objective function. This function is a measure of how close our approximation is to the data. There are various different choices that can be made, with increasing complexity and nuance as the problem domain becomes harder for computers. In the process of optimization we aim to reduce loss as close to 0 as possible. Here it will suffice to demonstrate one of the most common loss functions, the squared error: L = 1 2M X m (N m − y m) 2 (3.2) where y m is the target value found in data for m − th element of the data set, N m is corresponding output from neural network, M is the total number of data points 14 in the data set used to calculate L. This loss value acts as a guide to how well our model is doing in the process of optimization. As the loss decreases our model starts to resemble the data more accurately. Gradient descent is an optimization algorithm that relies on derivatives of the loss function with respect to the parameters of our model. If we label the parameters of our model as Wij basic structure of the gradient descent algorithm looks like this: Initialize eqn. 3.1 with random Wij while loss > desired loss do Compute derivative of L wrt parameters, ∂L ∂Wij Change the values of Wij to Wij − l ∂L ∂Wij end Algorithm 1: Basic structure of gradient descent Here l is called the learning rate. It controls how far we move in the direction opposite to the gradient in a single update step(gradient itself points in the direction of increasing L with respect to the parameters of our model). There are practical considerations in choosing l, two obvious cases being too large or too small values for l. In the former case this algorithm never converges and in the latter case it always converges but time to completion may be too long. Usually there’s a suitable intermediate value between large and small l. Common practice is to reduce the learning rate as training goes on, either automatically by the algorithm or by manually arranging its value. Learning rate is the first example of a class of variables called “hyperparameters”. The constants used in optimization algorithm, number of layers, number of neurons in the layers and many more belong to this category. In general any parameter that requires tuning and isn’t a weight in our neural network will be a hyperparameter. They can be assigned manually, or chosen through a “grid search” of several potential values. The method we use, namely hyperband, is explained later. There are various different modifications to the basic setup of gradient descent. These include adding a momentum term [14], adaptive methods that scale learning rate automatically [15] and each weight specifically [16]. All of them share the same underlying idea as the basic variant. 15 Backpropagation is the technique of moving the gradient of the loss function from output nodes towards input nodes. Since error of any node by itself doesn’t make sense except for the output, every parameter that eventually culminate in loss calculation gets judged by its contribution to the loss function. Below is backpropagation of errors in equation form for our 1 layer neural network in eqn 3.1, ∂L ∂Wij = 1 M X m (N m(Wij ) − y m) ∂N m(Wij ) ∂Wij (3.3a) ∂N m(Wij ) ∂Wij = aiσ0( X j (W1jXj + b1))Xj (3.3b) Here M is the total number of data points, with m denoting a single element of that set. σ0 denotes the derivative of the specific activation function used. It is because of this step that the activation functions in neural networks must be differentiable. With differentiable activations, calculating derivatives with backpropagation scales linearly with the number of data points. Doing the same task using the usual numerical differentiation methods would have resulted in O(n 2 ) scaling. This linear scaling with input, combined with parallel computing is what enables use of "big data". One final note regarding the use of gradient descent is that when passing through the data only a small portion of that data is seen at a single update step. This considerably speeds up the optimization process, even though each step is somewhat random. 3.4 Quality of Fit The procedure explained above is quite simple but in practice there are many pitfalls. Because neural networks contain a large number of parameters, it’s possible for them to overfit the data (“memorizing” data), fitting so closely that the model matches the noise in the data and end up generalizing poorly. It is also possible to have a network fail to extract all information from the data, which is called underfitting. These can be identified via observation of loss value over time during training. For purpose of testing generalization, a small subset of the total data called a “validation set” is used. The neural network never sees this portion during training, only 16 the loss value over this subset of data is calculated as well as the loss value over the training data (or training set). During the training these two values should decrease together. A decrease in loss value of training data while loss value of validation data is either increasing or remaining the same implies overfitting. The last iteration at which loss value of training data and validation data decreased together is the optimal time to stop training. Anytime before this would be underfitting, anytime after is overfitting. This method is called early stopping.[17] There are many other methods to control overfitting. Most straightforward option is just having more data, as it’s difficult to memorize all the training data when there is too much of it. Another method is called (weight) regularization, in which a term proportional to the weights or the square of the weights is added to the loss function. This penalizes using too many weights unnecessarily at the expense of a control constant, the regularization hyperparameter. This hyperparameter controls how strongly weights contribute to the total loss. Searching and using a value as close to optimal for weight regularization yields models that generalize with more accuracy. Another method is called dropout, where during training some random portion of the neurons in a layer are disabled by temporarily setting their activations to zero for a single iteration. This forces the network to use less of its available capacity at each iteration without sacrificing potential benefits from redundancy. The proportion of dropped neurons is another hyper-parameter, however it’s almost always optimal to use p=0.5 for dropout rate. [18] When training is finished, further tests of the model’s quality are done on a separate, third set of data called the test set. Since validation data is used to stop training at an optimal point, model is optimized over the validation set implicitly. There could be a configuration of parameters that give higher loss on validation set, but lower in a much more diverse data set. Those configurations aren’t chosen with early stopping, but the procedure identifies overfitting to training data and is a staple of the machine learning approach. Checking the quality of the final model can be done through many ways. While optimization procedure uses loss function (specifically mean squared error) as the metric of choice, any statistical metric is appropriate. These may be mean absolute 17 error, mean squared error, linear correlation coefficient, lin’s concordance correlation and many others. A way to probe the precision of the model is sampling initial values. Upon acquiring an initial weight configuration the network is trained until desired conditions are reached. This process is repeated with everything held exactly the same except the initial values. Several trained networks can then be used to estimate the standard deviation of the model. The causes that would contribute to such a variation are initial values and different local minima the network ends up in after training. Since the training and validation data, optimal stopping condition, network structure and optimization parameters are held constant, this amounts to estimating model uncertainty. It should also be noted here that in our case, the loss value is convex and as a result has a global minima which also is the only local minima. There are, however, many different configurations of weights that would yield results close to this value. The outlined method samples exactly these.