The deep neural network model of nonlinear regression of laminar combustion.

Rafał Lenartowicz

2017/04/26

Contents

1 Introduction.

The main goal of this work is to provide the model of a deep neural network which can, after learning on a particular data set, generalize its features and be able to find them on previously unseen data. Precisely, based on given data of laminar combustion process, the model must return values of laminar combustion velocity without knowing an analytic formula. Using a deep network gives an opportunity to find connections between wide scope of different kinds of parameters mixed together, which would be unobtainable with other methods. Even though this work focuses only on three-dimensional input, deep neural networks can be used on high dimensional space. Expected results must be consistent with experimental observation.

In the further parts of this paper sections about physical background of the investigated process, data preparation and architecture of moderns neural networks are included. The whole work is then summarized. In the end one may found list of papers used and some other resources.

2 Laminar combustion.

Laminar combustion refers to the combustion process in which there is no turbulence in a flame, e.g. flame of a candle. Laminar flame speed is a property of a combustible mixture. It is the speed at which a laminar flame will propagate through a quiescent mixture of unburned reactants.

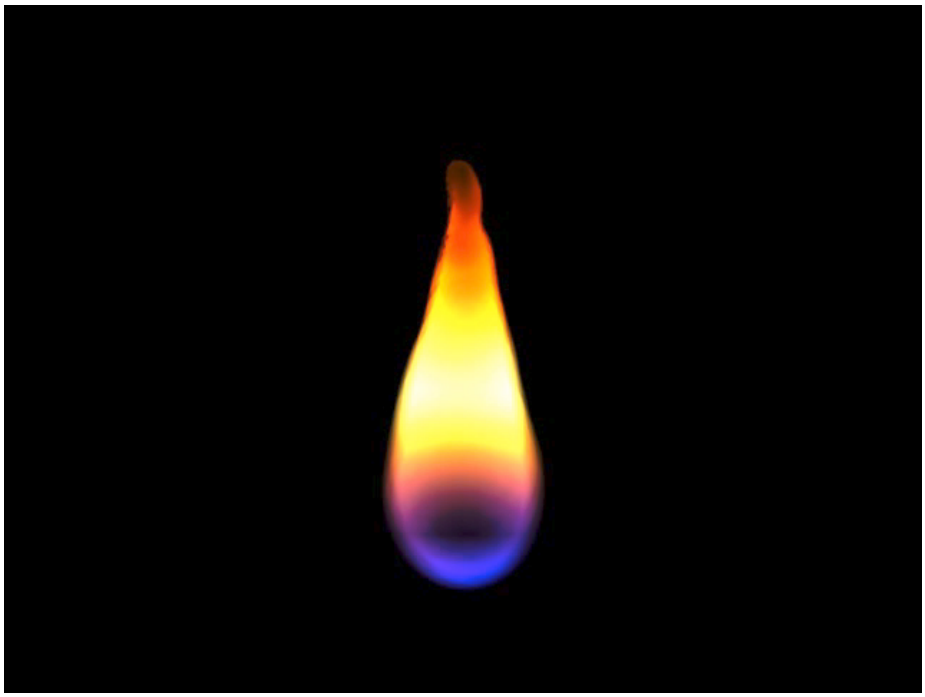

Figure 1: Laminar flame example.

3 Data analysis.

The given data set is a $10000x4$ matrix. It consists of four columns, of which in one there are values of velocity of flame v and in the other three are input parameters. These are:

1. temperature at which the process took place T [K];
2. pressure at which the process took place p [atm];
3. stoichiometry coefficient ϕ.

First, the whole dataset is inspected. It is shown in the plot below. Examining it, there are few features to be seen. The velocity changes in nonlinear way with ϕ. Then, there are ten groups of points, which share a common parameter, either T or p. Now, it is advisable to find how these two influence the output.

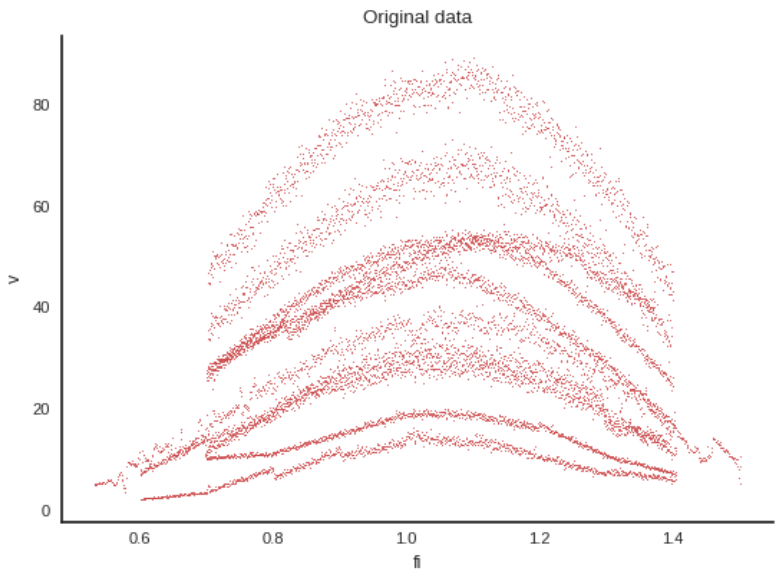

Figure 2: Whole dataset.

The temperature is shown increasingly on y axis. Pressure is constant for all three datasets. It is clear that velocity is proportional to temperature. Values of temperatures are accordingly: $300K, 343K$ and $478K$. Pressure is $1atm$.

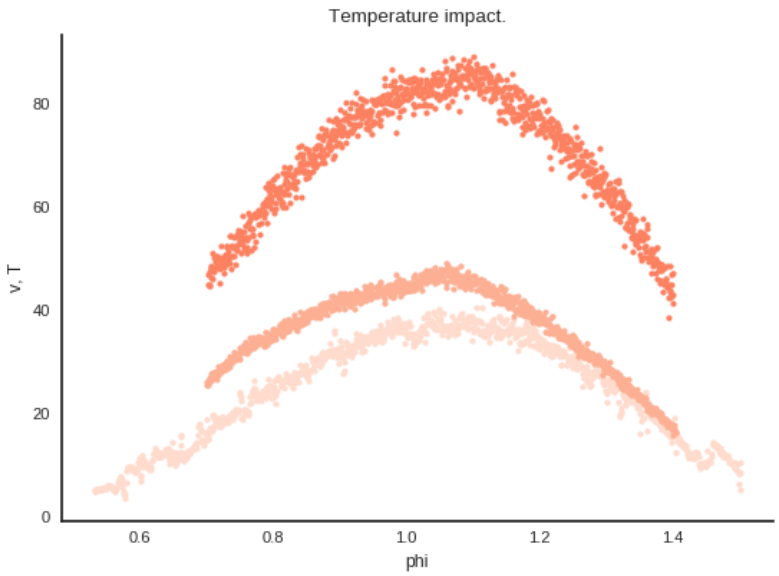

Figure 3: Temperature impact with constant pressure.

To inspect pressure influence, the temperature is constant ($300K$). Pressure values are respectively $1, 2, 5, 10$ atm. it must be noted that in this case growing pressure reduces velocity (pressure values are given by color saturation).

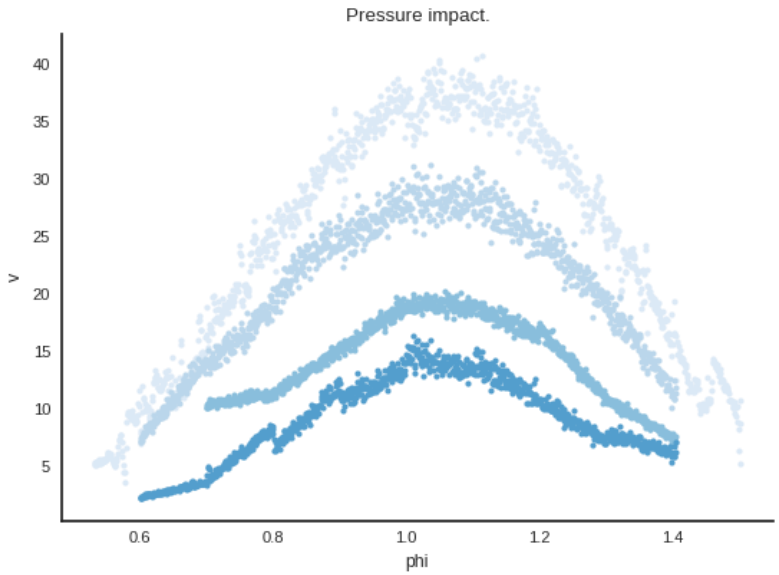

Figure 4: Pressure impact with constant temperature.

The last step is to prepare data to comparison. There are no missing values so therefore no clearing of NaN values is needed. Each column must be normalized separately, as they consist of values measured in different physical units. All values are positive, then it is reasonable to rescale all four vectors to $[0, 1]$ range. It would be done with the formula:

$$x_{normalized} = \frac{x - min(X)}{max(X) - min(X)}$$

where X is a vector of xs. Results are shown below.

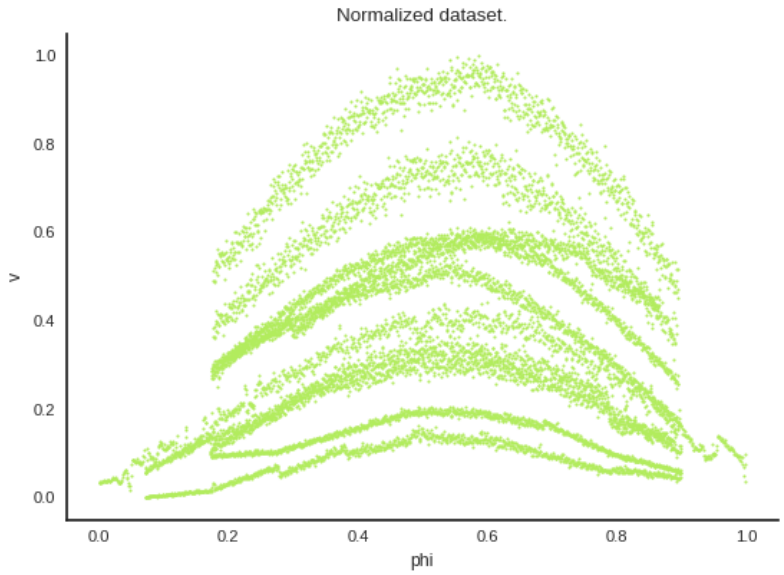

Figure 5: Normalized dataset.

4 Deep neural networks

4.1 General overview.

Deep neural networks are a set of algorithms which are fed with some input, transform it in some way and compare results with what should be obtained (this is an example of supervised learning. There are others types in general, but those do not apply to the examine case). Then the errors are calculated - to judge how far from exact solution a model is. The error is passed back [1] to adjust weights and the whole process continues until reaching either the end of inputs or satisfactory level of performance. Even though a neural network with only one hidden layer can approximate any function as good as it is wanted [2], deeper networks seem to be better [3].

Here three deep layers are used. For this particular case the network presents better results with wider rather than deeper architecture. In particular, this set of layers was used to build the network:

1. Input layer with 150 neurons.
2. Deep layer with 3 000 neurons.
3. Deep layer with 3 000 neurons.
4. Deep layer with 500 neurons.
5. Output layer with one neuron (as the expected output is a single value).

4.2 Mathematical model.

A brief description of how neural networks work is given below, as well as an exemplar image.

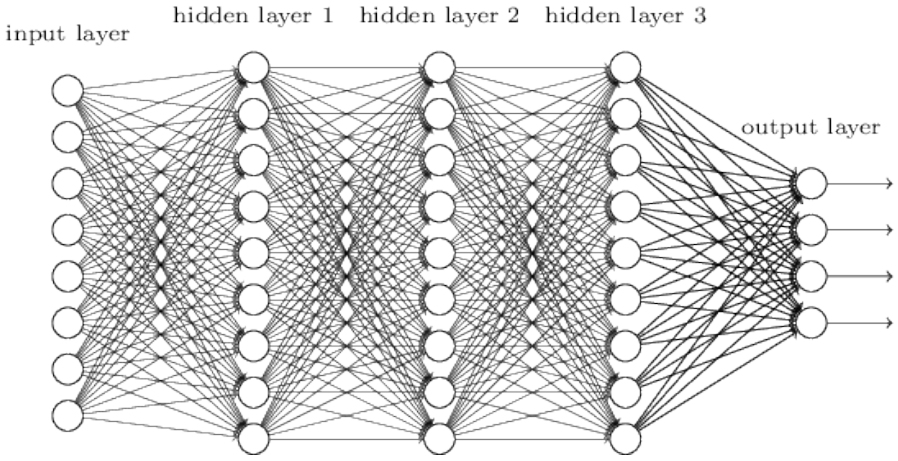

Figure 6: Deep neural network schema.

Let's consider a jth neuron from a llt layer. Then:

1. ${y^l}_j$ is an output of jth neuron from a llt layer. It is send to all of neurons in $l+1$ layer, but to none of its neighbors. Neurons are only connected between layers.
2. ${w^l}_k$ is a weight applied to an output from the kth neuron from the $l-1$ layer.
3. ${a^{l-1}}_k$ is the kth output from the $l-1$ layer.
4. b^l is bias of the lth layer.
5. σ is an activation function.

With this one can write the equation describing any neuron. First, it multiplies a weight and an output from previous layer:

$$ {w^l}_k \cdot {a^{l-1}}_k $$

Because it is connected to all of the neurons in the layer above it, e.g. m neurons, it sums m multiplied pairs:

$$ \sum_{k=1}^{m} {w^l}_k \cdot {a^{l-1}}_k $$

To this the bias is added (bias is very important, as it allows a neuron to produce values unobtainable otherwise).

$$\sum_{k=1}^{m} {w^l}_k \cdot {a^{l-1}}_k + b^l$$

In the end, an activation function is applied:

$$\sigma(\sum_{k=1}^{m} {w^l}_k \cdot {a^{l-1}}_k + b^l)$$

The output of the neuron is therefore:

$${y^l}_j = \sigma(\sum_{k=1}^{m} {w^l}_k \cdot {a^{l-1}}_k + b^l)$$

Wrapping this to the vector form one obtains:

$$Y^l = \sigma(W^l \cdot Y^{l-1} + B^l)$$

When an input vector passes through all of layers, output from the last layer becomes an output from the whole network. Depending on what one want to get, there are many way to manipulate the network. In this case there should be just a single numeric value at the end of the network, additionally in range $[0, 1]$. To assure this, sigmoid activation is used and there is only one neuron in the last layer.

While training, the error is calculated (i.e. loss function). Some common formulas can be used, like squared error, logloss, quadratic cost function etc. Here the mean squared error is applied. Then, there is calculated gradient and it is propagated backward, so the network can adjust better values for its weights. Biases are also updated.

4.3 Activation function.

An activation function is necessary to input nonlinearity into a model. Although basically one can use almost any function, some are more accurate. The common activation functions were e.g. sigmoid or tanh. Here the ReLU (Rectified Linear Unit)function is used [4]. It is simply yet well performing function:

$$f(x) = \begin{cases} 0 & if \quad x < 0 \\ x & otherwise \end{cases} = \max(0, x)$$

4.4 Kernel activation.

The first values of weights must be initialize somehow. Many algorithm are more stable when both inputs and weights are close to zero. Small inputs are assured by normalization of data. The kernel activation method generates tensor with a normal distribution, $\mu = 0$ and $\sigma = 0.05$. After the first step of back propagation the weights are adjusted to better fit the model.

4.5 Optimization algorithm.

The reason to optimize back propagation step is simply to save time and computational power. In classic gradient descent algorithm one has to calculate all of the errors, but it is better to calculate just some of them and do a smaller step toward the optimum. The points are chosen randomly what gives a name to the stochastic gradient descent [5].

In the network in this project the ADAM [6] optimizer has been chosen. Describing this algorithm is beyond the scope of this work. It will be just mention that it is computationally efficient and applicable to many deep learning problems.

4.6 Dropout.

Dropout is a very popular method to avoid overfitting [7]. The smallest dataset, the bigger the chance of overfitting. It means that a big network trained on small amount of inputs becomes blind if it has to perform on new, yet unseen data. It is so because there is not enough example and weights are only sensitive for known values.

There is also one additional reason. It may occur that neurons depend too much on some neighbors. It is strongly unwanted, because such pairs can learn how to correct each other mistake, but they will not be able to act properly on new input. Dropout is a very simple method to prevent that.

Using a 50 % dropout after each layer means that which each batch of data, half of neurons are inactive. When using trained network on real data, all neurons are used. Instead of dropping out some of them, all the weights are reduced proportionally. Neurons are chosen randomly, so it is no longer possible for two neurons to depend too much on each other. It also acts like dividing a network into many smaller networks, which are then merged. It is generally better to have many networks and to merge them rather than have only one, even bigger. Dropout technique allows to obtain it without more work nor computational power.

5 Results.

Having 10 000 rows, it was decided to divide the data into train and test sets. They have 7 000 rows and 3 000 rows respectively. On the plot below blue points represents values never seen by the network. Red points are the network predictions.

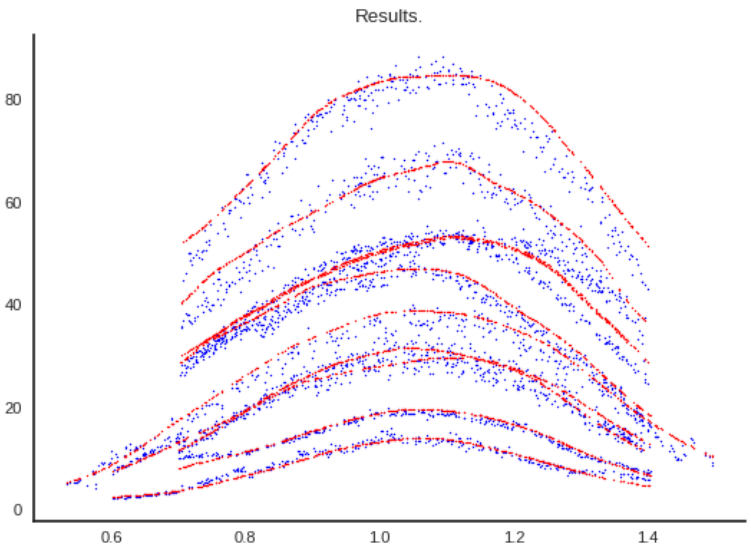

Figure 7: Blue points were to be predicted, red ones are the network outputs.

As one can see, the network generalize over the dispersion of the original data. It rather returned central lines of sets. It also created ten lines, so it is plausible to claim that the network was able to exactly distinguish between different series.

The main drawback of the dataset is low variation of temperature and pressure values, yet they have to be treated as continuous rather than discrete variables.

The network had also trouble with correct recognition of two different sets while there were mixed together for some ϕ values. One can notice it while looking at middle right part of the plot.

The mean squared error is 5.5204887. Even though the network has managed to find general rules and estimate the process, it needs some tuning. It would be advisable to use more data to train it, especially with wider range of temperature and pressure values.

6 Resources.

References

[1] David E. Rumelhart, Geoffrey E. Hinton†, Ronald J. Williams (1986) *Learning representations by back-propagating errors*

[2] G. Cybenko (1989) *Approximation by Superpositions of a Sigmoidal Function.*

[3] Razvan Pascanu et al. (2014) *On the number of response regions of deep feedforward networks with piecewise linear activations*

[4] Vinod Nair, Geoffrey E. Hinton (2010) *Rectified Linear Units Improve Restricted Boltzmann Machines*

[5] Stephan Mandt, Matthew D. Hoffman, David M. Blei (2017) *Stochastic Gradient Descent as Approximate Bayesian Inference*

[6] Diederik P. Kingma, Jimmy Ba (2014) *Adam: A Method for Stochastic Optimization*

[7] G. E. Hinton et al. (2014) *Dropout: a simple way to prevent neural networks from overfitting*

[8] Trevor Hastie, Robert Tibshirani, Jerome Friedman *The Elements of Statistical Learning*
`http://statweb.stanford.edu/~tibs/ElemStatLearn/printings/ESLII_print10.pdf`

`http://machinelearningmastery.com/`
`http://cs231n.github.io/`
`https://keras.io/`