Rafał Lenartowicz 29.06.2017

WARSAW UNIVERSITY OF TECHNOLOGY

Division of Aircraft Engines

DEEP NEURAL NETWORKS
Predictions of velocity of laminar flame.

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.

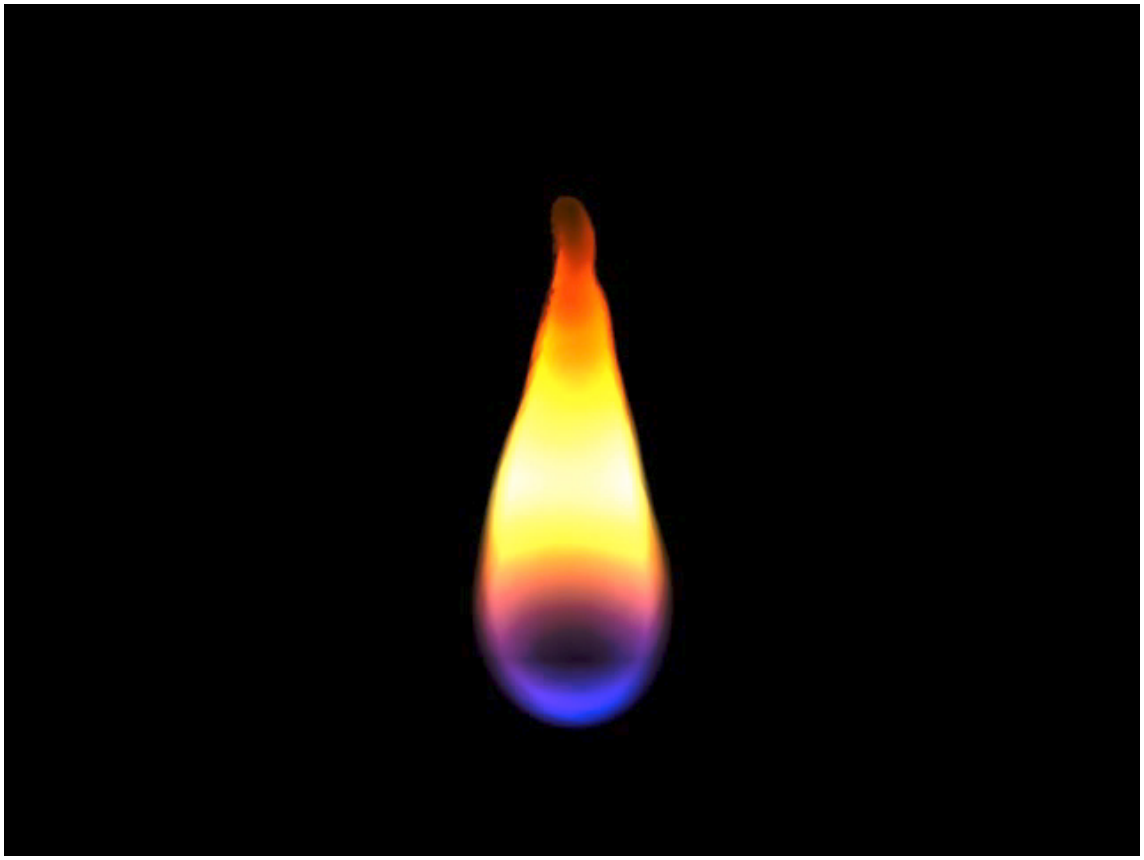

Figure 1: Laminar flame example.

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.

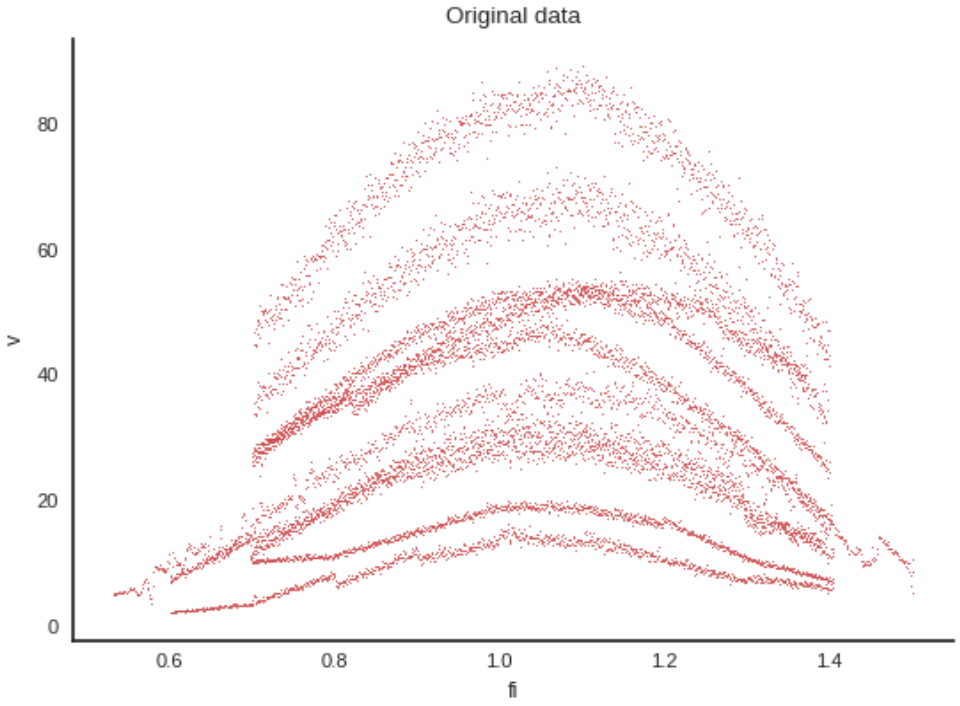

Figure 2: Whole dataset.

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 $1atm$.

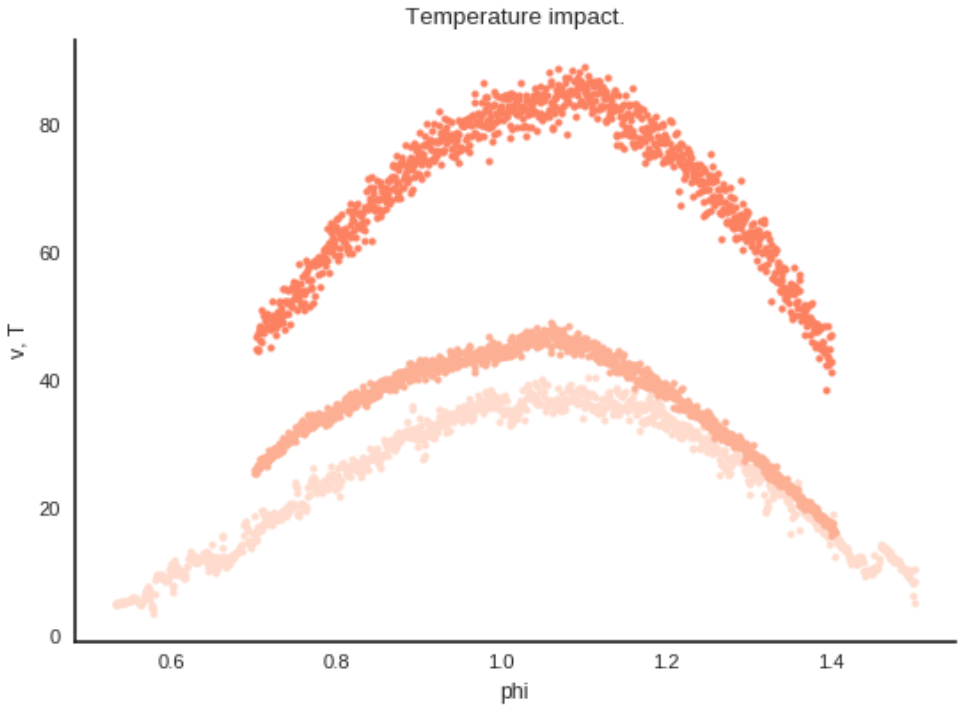

Figure 3: Temperature impact with constant pressure.

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).

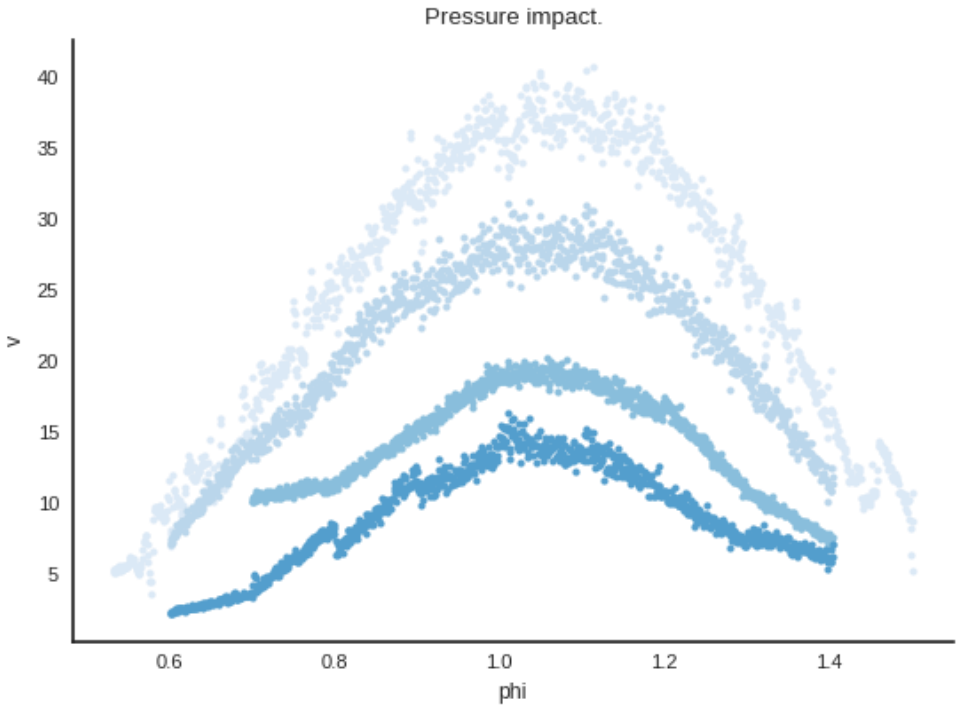

Figure 4: Pressure impact with constant temperature.

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. Important note here is that data set must be split into train and test sets first due to possible information leakage. As one can see, randomly sampling test data it is not necessary that max and min values of both sets will be equal. Thus, the normalization process will differ.

Results are shown below.

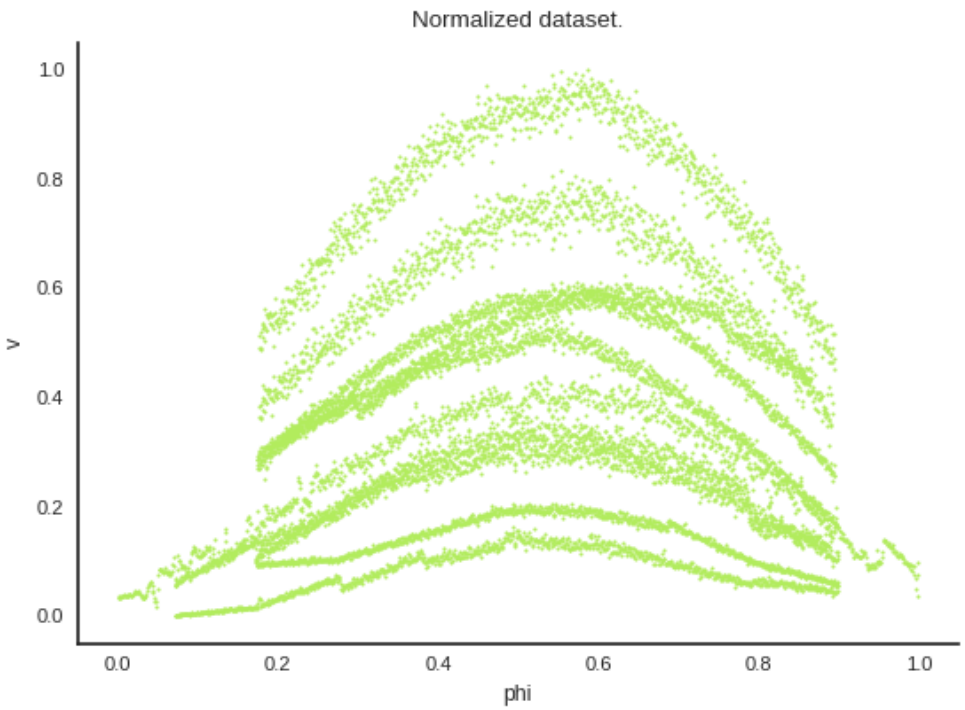

Figure 5: Normalized dataset.

4 Dense 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].

4.2 Densely connected layers.

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).

4.3 Mathematical model.

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

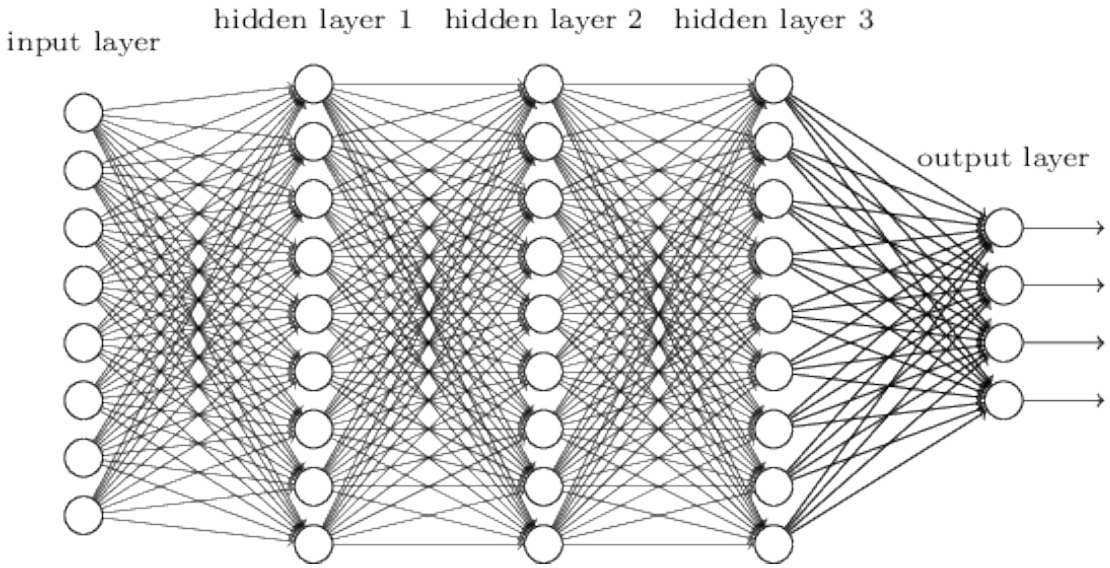

Figure 6: Densely connected neural network schema.

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.

4.4 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:

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

4.5 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.

4.6 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.

4.7 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.

4.8 Alpha dropout.

This kind of dropout technique is in general similar to the one described above. Alpha dropout [8] is a dropout that keeps mean and variance of inputs to their original values, in order to ensure the self-normalizing property even after this dropout. Alpha Dropout fits well to Scaled Exponential Linear Units (SELU) by randomly setting activations to the negative saturation value. It is not used here as RELU activation method has been chosen.

4.9 Batch Normalization.

Sometimes output of a hidden layer may reach huge, and at the same time numerically unstable, values. It has negative impact on a model, because both back propagation of error is devastating for smaller weights and many neurons can simply play no role, if their output is smaller by the orders of magnitude. This may be soften by lowering a learning rate, but at a cost of much longer computation. Another solution, used in the project, is process of batch normalization [9]. Batch normalization is nothing more but normalization process (similar to the one described above), but this time output of each layer is rescaled. Due to the cited paper, batch normalization process yields great results and can be applied instead of dropout, or at least parallel to it.

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.

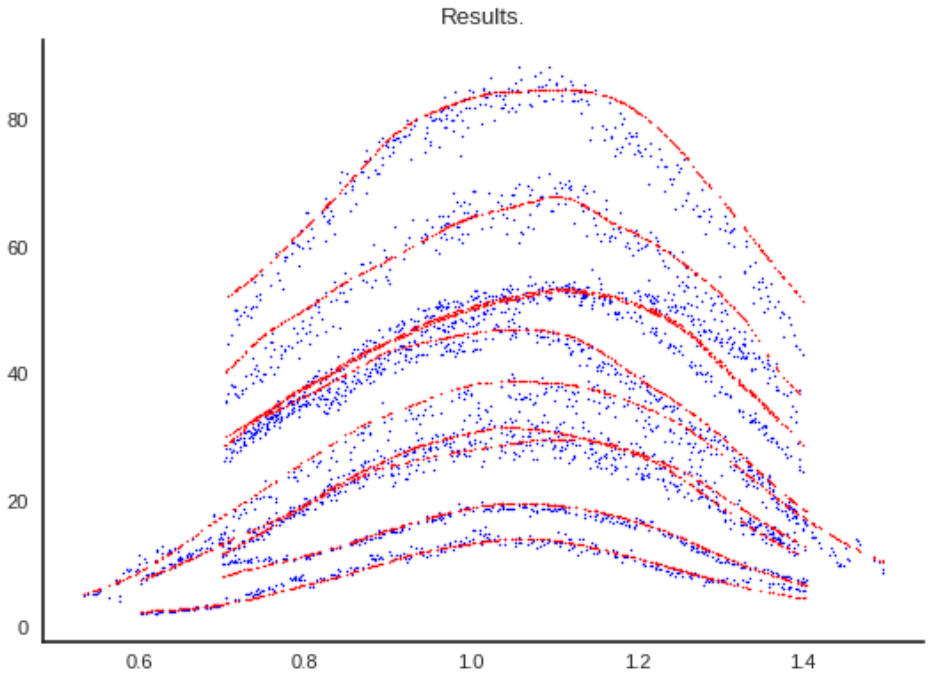

Figure 7: Blue points were to be predicted, red ones are the network outputs.

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 Different approach - time series.

As shown earlier, the problem can be solved as a classic regression problem. There is yet another possible solution. Suppose that the function

$$v = f(\phi, T, p)$$

can be treated as time series, that is - ϕ has now a direction. It starts from lowest value and increases at constant ratio, just like time. Then, v is a function of time-likish variable ϕ and some two other variables. In this approach a network will be given all values from start to some point to train at. Then, feeding it with only independent variables of the rest of the data, the network is supposed to return predicted values of v. This can be done using different types of network, of which two will be used here: LSTM and RNN.

7 RNN - recurrent neural networks.

7.1 Description.

The idea behind RNNs is to make use of sequential information. In a traditional neural network it is assumed that all inputs and outputs are independent of each other. Even though it works, what is proved above, it is not necessary true. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a memory which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps, which should be enough for this experiment.

This type of network can be used to learn extracting words meaning or, as in this particular case, finding a pattern in a sequential data.

To give some general idea of how this network is designed, one may look at the picture below.

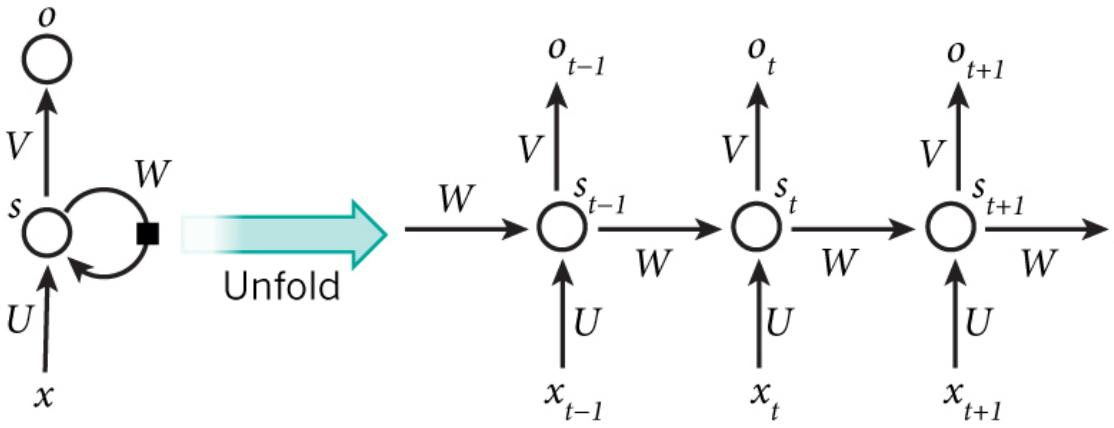

Figure 8: Recurrent neural network schema.

Unfolding means creating the very same layer for each time step of data. Calculated output is then feed to the very same network, but some values may be preserved. It is represented by s_i values, which can be described as memory cells. Their values are computed based on the previous hidden value s_{t-1} and current input x_t. Here, t means some value of time variable. x_{t-1} is an input vector from same series, but coming before the x_t. It continues with x_{t+1}, which is an input vector at the later time. Following outputs o_i are fed to the network as long as the current series last. The last output is the general output of the network.
U, W, V are parameters of a network, as described in the DCNN section. Here they are shared between layers, which reduces their total number. The s value, the memory, should be collecting infomation about what has been happening in previous steps, but it is greatly limited and cannot remember too much from the past.

resources: [10]

7.2 Issues.

As mentioned earlier, the RNN cannot remember trend for too many samples. In this case, where one series has 1 000 steps, the RNN would be inefficient compare to different network with the same size. The better architecture will be used, as described in the next chapter.

8 LSTM - long short-term memory networks.

8.1 Description.

Long short-term memory networks are very similar to RNNs, but more complicated and considered better. The RNNs have memory cells, which are often refer as gates. The LSTMs have more gates - three. Additional to the memory gate, there are write gate and read gate. The LSTM unit consists of a memory cell which attempts to store information for extended periods of time. Access to this memory cell is protected by specialized gate neurons - the keep, write, and read gates - which are all logistic units. These gate cells, instead of sending their activities as inputs to other neurons, set the weights on edges connecting the rest of the neural net to the memory cell. The memory cell is a linear neuron that has a connection to itself. When the keep gate is turned on (with an activity of 1), the self connection has weight one and the memory cell writes its contents into itself. When the keep gate outputs a zero, the memory cell forgets its previous contents. The write gate allows the rest of the neural net to write into the memory cell when it outputs a 1 while the read gate allows the rest of the neural net to read from the memory cell when it outputs a 1. Thanks to this solution, the problem of exploding or vanishing gradient, known from RNNs, can be solved. One can see unfolded LSTM unit below:

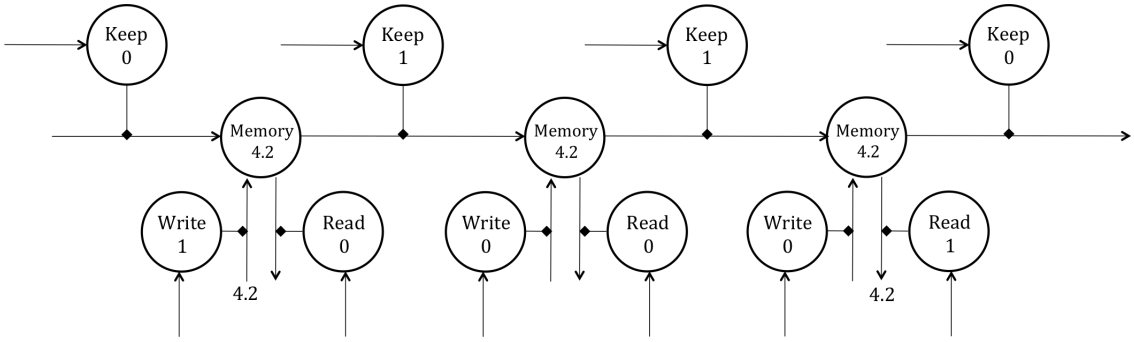

Figure 9: Long short-term memory neural network schema.

At the first step keep gate is 0 and write gate is 1, which forces memory cell to remember given value (here 4.2). The datum is then preserved for one step (thanks to keep value 1). In the last step the datum is read. This process lets remembering data for as long as wanted and deal with unwanted gradient values.

resources: [11]

8.2 Neural Turing Machines.

Yet another improvement to RNN has been created and named Neural Turing Machines [14]. The main difference is that in this model, memory cells are not hidden but rather kept externally. The model is not used in this experiment as it brings nothing new to the solution of the problem.

8.3 Data preprocessing.

The data need some additional preparation. First of all, only one set from dataset is chosen. This reduces the dimensions to two only, as both p and T are constant. The set has a size of $[1000x2]$. This time the data is normalize to $[-1, 1]$ values by:

$$x'_i = 2 \cdot \frac{x_i - max}{max - min} - 1$$

The next step is to make data stationary. It is done by following formula:

$$x'_i = x_i - x_{i-1}$$

By subtraction previous value from current one only information about the change is left. This leads to the notion that steps are distributed equally and step value is constant, thus they can be deleted from the experiment.

In the end, the problem is reduced to univariable. To transform it into supervised learning case, a network should be fed with value from all past steps and asked for a value in the next one.

The following LSTM architecture is used:

1. number of neurons = 8
2. number of epochs = 500
3. batch size = 1 (the network moves one step forward at each epoch)

This is relatively small network. Bigger networks were reporting worse results as the data is very limited. in general LSTM networks require more computational power, but this was not a case here.

The experiment produces three kinds of results which are commented and illustrated below.

8.4 Local trends.

The 70% of the data without trend is shown to the network. Then the network is asked to predict the following 30%. The plot shows that real values (green line) is mimic by the network output (reddish line), but none of extreme values is reached. Mean squared error is $r = 0.67$. The overall result is plausible.

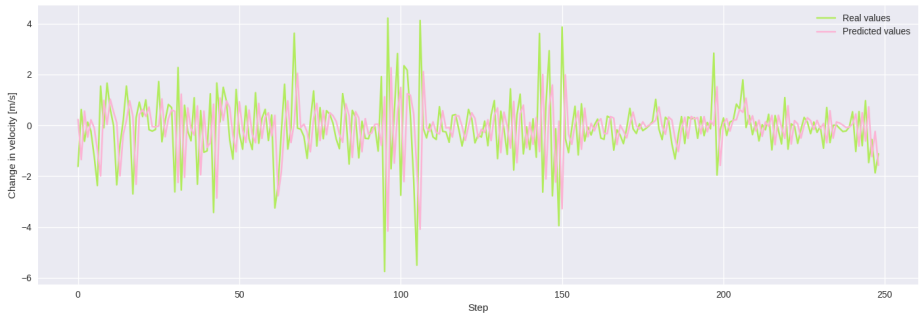

Figure 10: Plot of predictions of local fluctuations of velocity.

8.5 Trend line.

The next plot presents results for non stationary data. One can see that the network was not able to catch the proper angle of the line. The error here is rescaled by both trend and reversed normalization: $r = 17.45$. This result is not not satisfactory.

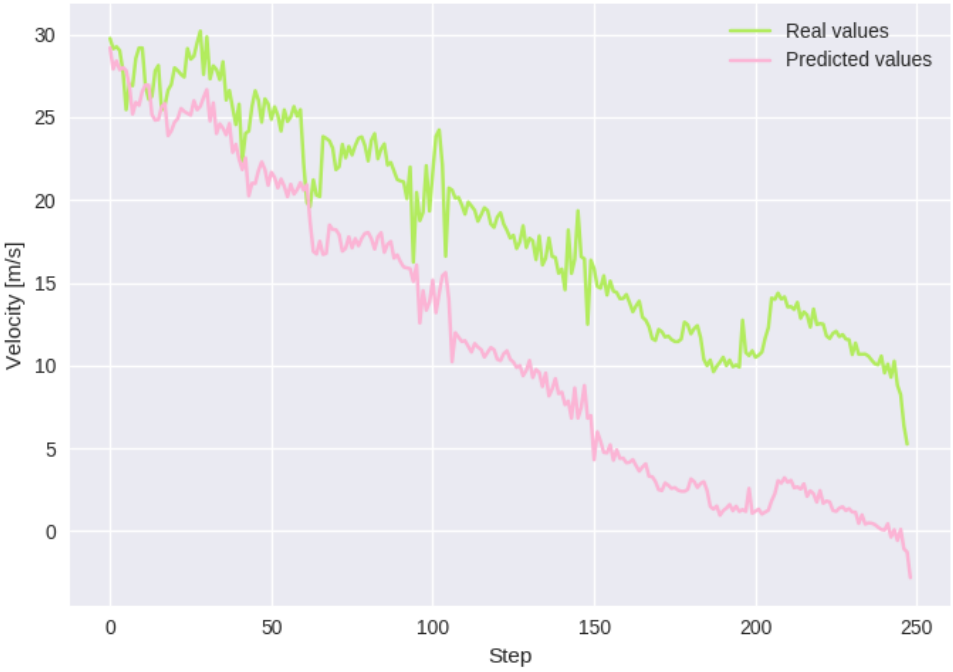

Figure 11: Predictions of a trend line.

8.6 Final results.

Here the whole output is shown as well as original dataset. The error of generalization over the proper angle is still visible.

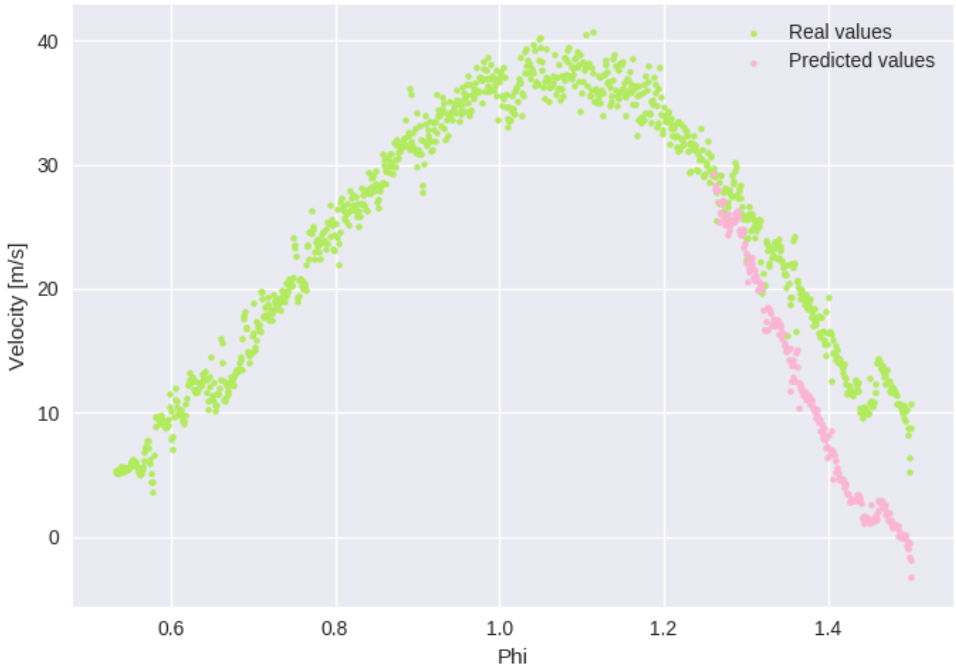

Figure 12: Final results.

The plot above shows what was deduced from earlier ones. The network was able to predict the shape of a line with acceptable accuracy, but it failed in keeping the track of the trend.

In comparison to DNN, LSTM network presents worse performance. Because these two methods of solving the problem are different from one another, only a holistic overview is possible to present. It must be mention that LSTM network consumed about 100 times less resources (time, computational power), but was dealing with only 10% of the dataset. The network itself was smaller in every dimension. In general, LSTM networks require more resources as they are more complex.

9 Convolutional Neural Networks.

The last type of network to be described. It is not going to be used as a possible solution due to a nature of the algorithm. CNNs were the revolution in image processing. They are capable of extracting many, both low and high level features from given picture. There are used to find objects in photos, read text and numbers, guess feelings from a face image etc.

9.1 Overview.

Convolutional Neural Networks are very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function on the last (fully-connected) layer.

CNN architectures make the explicit assumption that the inputs are images, which allows to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

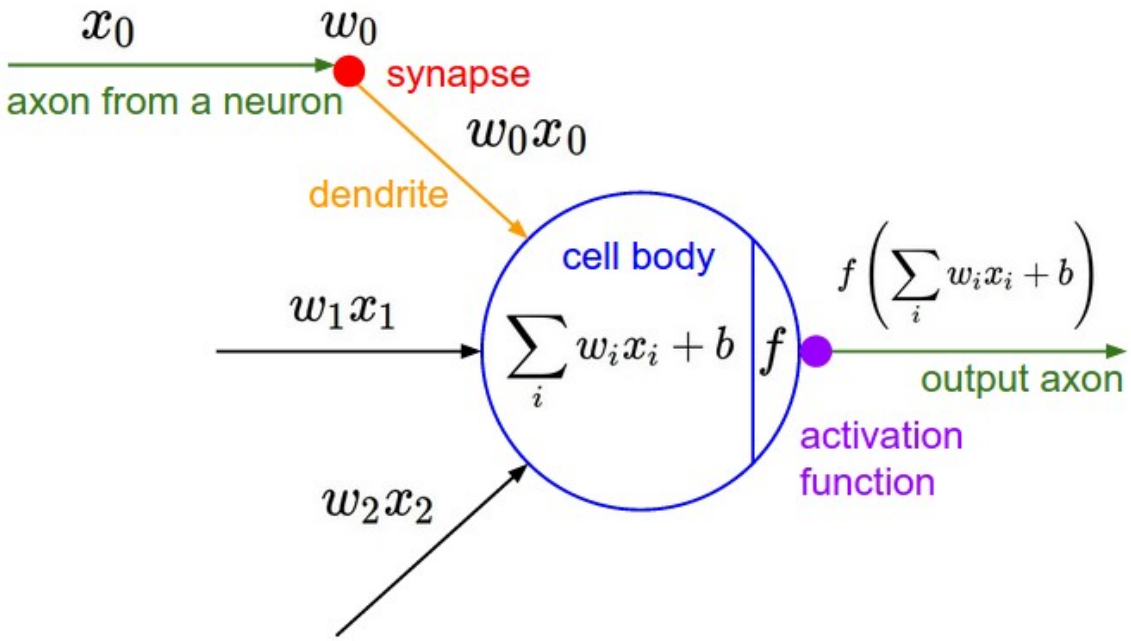

Figure 13: Convolutional neural network schema.

The general idea is still the same as in the Dense-connected neural networks. There are weights w_i, output from earlier layer x_i, activation function and output.

The difference lays in convolution process. In mathematics, convolution is defined as follow:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

and it produces a third function given two functions. In CNNs an image, which is a 3D set of pixels, which are but numbers from $0, 1, ..., 255$ domain, is scanned with a filter. A filter is an imaginary square, which can read some, generally few, like 9 or 16, pixels' values at a time. It then applies some function to all these values, most of the time it is simply a sum. The result is being passed deeper into a network and the filter moves by given value to read the next set of pixels. It continues until it has scanned the whole image. The weights which are applied to each filter, as a network commonly consists of many of them, are to be learnt.

To generalize and extract objects CNNs have to reduce the shape of harvested data. A common example is to take only a highest value or a mean value from each four (2x2 square) filter's outputs. It is shown in the picture below:

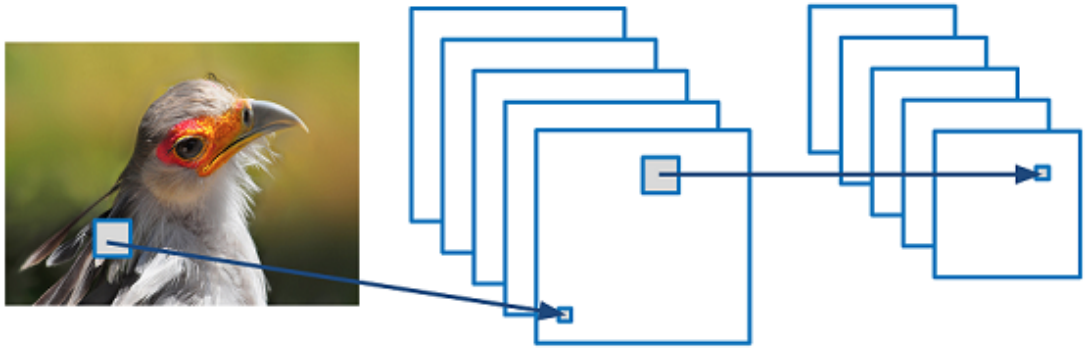

Figure 14: Filtering in CNN schema.

resources: [12], [13]

10 Additional auxiliary methods.

There are some mechanisms to boost accuracy of networks. These include better data handling and using multiple types of algorithms rather than just one.

10.1 Ensembling

Ensembling is the method of boosting networks of neural networks in many ways, letting obtaining results above the human capability [15], [16]. It is not used here as there is the very small dataset and no need for further push of accuracy.

It is worth mentioning though, as ensembling plays the core role in all of deep learning tournaments and researches. There are many ways one can do it, but the general idea is simply to extract as much information from given as possible. To give an example, in a simply classification problem, one can train three models with as high variance as possible. Then, using e.g. democratic voting, noticeably improve the final accuracy. It is because, if each model can predict good result with 70% probability, then:

1. all three models are right:
$$0.7 * 0.7 * 0.7 = 0.3429$$
2. two models are right:
$$0.7 * 0.7 * 0.3 + 0.7 * 0.3 * 0.7 + 0.3 * 0.7 * 0.7 = 0.4409$$

So, simply adding this up, the general accuracy of the merged model will be

$$0.3429 + 0.4409 \approx 78\%$$

which is significantly better than single 70%. In most cases models which know the true vote right as majority.

As mentioned, there is much more methods, most of them are also more sophisticated [17]. The biggest disadvantage of ensembling is size of all networks and models stacked up. In many cases a huge architecture must be reduced to obtain results in reasonable time. It is also import to have as different and uncorrelated models as possible.

10.2 Data augmentation.

One of the most important statement is the more data the better. It is possible to artificially generate data to improve overall capability of a network. Most of the listed methods are not suitable for all of the cases, but are promising in the other.

1. One of the most common technique is resampling. Resampling should be done in both directions. Categories which have highest population should be reduced and rare categories should be duplicated. The simplest method would be copying data from original dataset. It may results in lowering the variance and in the end bring no improvement.
   Resampling must be done carefully as it can shift some meaning from the data. Frequency of events most often helds information about an inspected phenomenon.

2. Other possibility is to change order in some sequences of data, like e.g. words order in a sentence. This operation is not always applicable, but worth mentioning. By doing this the general content of data is unchanged (if language is being processed as e.g. bag of words), but it can be treated as new datum.

3. One can stack some input lines into a square matrix and then, using CNNs, process it like an image. This can yield positive results as some more general features might be extracted.

4. Adding white noise is another method. It helps increase variance of data, what leads to train more robust network.

11 Summary.

As proved, there are many different types of neural networks which can obtain fine results on such class of problems.

There are some final statements which deserve highlighting:

1. data cleaning and preparation play very important role;
2. using computationally expensive methods like LSTMs on such kind of problems is highly inefficient and should probably be avoided;
3. neural networks generally perform better than other machine learning methods on much larger datasets;
4. stacking different models of networks and merging them can yield results unobtainable with any other methods;
5. albeit powerful, neural networks need parameter tuning and thoughtful selection of optimization algorithms;
6. while there are many current researches in the field, progress in deep learning has been driven by increasing computation power;
7. some actions should be taken to enlarge and/or improve data used to train and test a netowrk.

12 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] Sepp Hochreiter et al. (2017) *Self-Normalizing Neural Networks*

[9] Sergey Ioffe and Christian Szegedy (2015) *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*

[10] `http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/`

[11] `http://nikhilbuduma.com/2015/01/11/a-deep-dive-into-recurrent-neural-networks/`

[12] `http://cs231n.github.io/convolutional-networks/`

[13] `https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner27s-Guide-To-Understanding-Conv`

[14] Alex Graves et al. (2014) *Neural Turing Machines*

[15] `http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-image`

[16] Andreas Toscher and Michael Jahrer (2009) *The BigChaos Solution to the Netflix Grand Prize*

[17] `https://mlwave.com/kaggle-ensembling-guide/`

[18] 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/`