Learning a Poisson Equation

Verena Christina Horak

Institute for Mathematics
and Scientific Computing
University of Graz, Austria

1 Formulation of the problem

In physics, as in other natural sciences, it is of utmost importance to solve partial differential equations (PDE). Due to the rise of machine learning techniques it seems natural to learn PDEs with neural networks. The focus of this report lies on the following Poisson equation:

$$\begin{aligned} -\operatorname{div}(q\nabla u) &= f \qquad \text{in } \Omega \\ u &= 0 \qquad \text{on } \partial\Omega. \end{aligned}$$

For reasons of simplicity, it is further assumed, that $f = 1$ everywhere and q is a function that takes on values in $[1, 2]$ on Ω. Ω is represented as a 16×16 grid. As it is common in natural sciences as well as in the medical domain (i.e., MRI scans) that the actual u can be measured at least partially, it is assumed that u is known on Ω and the Poisson equation should be solved for q, which might be, for example, a parameter related to a specific tpye of tissue as seen in an MRI scan.

1.1 Generation of training data

One of the first things that has to be considered is the generation of training data. For this purpose, a large number of (u, q)-pairs was generated by a MATLAB code of the structure of Algorithm 1. This code generates qs in the form of the beginning of a cosine based series development with a total of 9 random variables as coefficients. To ensure that every q only takes on values on the closed interval $[1, 2]$, a final scaling is done. For every such q the Poisson equation is solved for u with a 5-point-stencil. These us constitute the input of the neuronal nets presented in the next section, whereas the qs are the labels (the output) that should be learned.

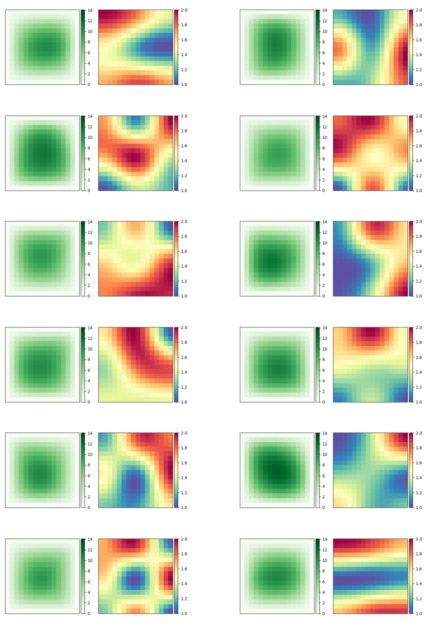


Figure 1: Some examples of (u, q)-pairs generated as indicated in Algorithm 1.

To give an impression of the difficulty of learning this Poisson equation by only knowing such (u, q)-pairs, Figure 1 should demonstrate how homogeneous the input data – a proper subset of all this us – and how different the output data (labels) – the corresponding q to a given u – are.

Algorithm 1: Generation of a (u,q)-pair

```
A = 0 and f = 1 everywhere ($16 \times 16$);
[X,Y] = meshgrid((0:15)/16*pi);
for k=0:2 do
    for l=0:2 do
        A = A + randn(1)*cos(k*X).*cos(l*Y);
q = 1 + (A - min(A(:)))/(max(A(:)) - min(A(:)));

u ← solve Poisson equation for this q;
```

2 The model evolution

The models described in this chapter were all realized in Keras (version: 2.1.3) on top of Tensorflow (version: 1.5.0) and the results were computed on a graphical processing unit (GPU) with the following specifications:

- Producer: NVIDIA Corporation
- Product: GeForce GTX 750 Ti
- CUDA Cores: 640
- Base Clock: 1020 MHz
- Memory Clock: 5.4 Gbps
- Standard Memory Config: 2048 MB

If not mentioned otherwise, Adam [KB14] with its suggested standard learning rate of 0.001 was chosen as optimizer, relu (Rectified Linear Unit) as activation, glorot_uniform [GB10] as initializer, and mean squared error (mse) as loss function.

2.1 Dense layers...

To get a first impression of how a sequential neuronal network should be designed in order to be able to learn the Poisson equation as described above, it was very useful to play around with just a few dense layers of different sizes. After some experiments, it seemed to be a good choice using four times as many neurons for these dense layers as the number of input values and only two (for **Dense 1**) or three (for **Dense 2**) such layers together with a dropout layer and the final output layer of same size as the output. These two nets together with the losses and validation losses are depicted in Figure 2.

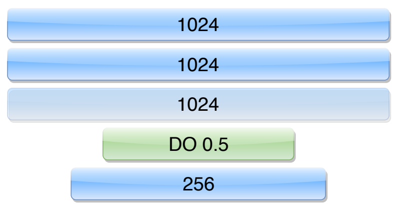


Figure 2: **Dense 1**: loss=0.0031, valLoss=0.0012
Dense 2: loss=0.0016, valLoss=1.1 e-4

In this figure as well as in all other figures describing neuronal networks, the layers are displayed from top to bottom. Furthermore, dense layers are colored in blue and the number of neurons of each layer is printed inside the bar, whereas dropout layers are indicated by green bars as well as the abbreviation "DO" followed by the dropout rate.

Dropout layers in general deactivate a certain percentage of the layers neurons and therefore constitute a regularization tool that helps to avoid over-fitting.

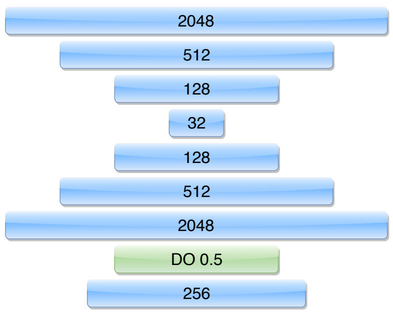


Figure 3: **Dense 5**: loss=7.9 e-4, valLoss=4.0 e-5

The next real improvement was reached by introducing a so-called encoder-decoder architecture, i.e., a net for which the dense layers resemble the shape of an hourglass. Whereas the upper triangular seems to guarantee a faster convergence behavior and a lower training and validation loss, the lower triangle helps to propagate the necessary information to all neurons in the output layer. This narrowing of the whole network can be interpreted as a possibility to "focus" on the most important data on which the output is based. The configuration of layers and their associated neurons for **Dense 5**[1] was determined empirically. Using less than 32 neurons in the narrowest layer in the middle of the hourglass resulted in over-fitting. All other changes of the number of layers or number of neurons that were tested just led to worse results than those stated in Figure 3.

2.2 . . . and convolutional layers

Although the results gained from Dense 5 are already remarkable, it might seem strange that the originally 2-dimensional data (a 16×16 grid) is fed in as a 1-dimensional input vector of length 256, although the generation of the data ensures that there is some neighborhood connection.

To benefit from this a priori knowledge, the upper triangle of Dense 5 was replaced by a combination of layers that is well-established in the context of image processing and helps to gather information from locally close neighbors. Typically, two blocks of 32 and 64 convolutional filters of a certain smaller size, i.e., 2×2, are generated, their results are gathered by a so-called max-polling which takes the maximum value of a specified rectangle.

[1]The names of the networks describe the type of the initial layers and indicate the progress of the project. "Dense 5" thus indicates that there were two network development steps (Dense 3 and Dense 4) which are not presented here since Dense 5 contains everything which was learned from them.

Within the context of this report, it was determined empirically that a total of 8 and 16 of such convolutional filters, with a kernel size of 2×2 and a striding of 2 in both directions, gives the best results. Here, a striding of 2 means that not the next neighbors are taken into account but the one after the next. To be able to continue with the lower triangle of Dense 5, the 2-dimensional result gained from the second of these max-pooling layers (green bars) needs to be flattened, which is indicated by a gray bar. The structure of this network **Conv 8** is depicted in Figure 4, which displays the convolutional layers as orange bars. As explained in footnote 1, "Conv 8" indicates that there were seven other networks with convolutional layers before – each best of its time. But only the combination of the convolutional block and the lower dense triangle resulted in remarkable improvements compared to Dense 5.

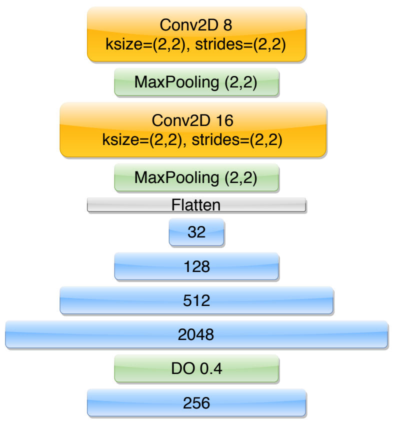


Figure 4: **Conv 8**: loss=5.7 e-4, valLoss=1.8 e-5

After playing around with this type of network, the impression was gained that Conv 8 is already the best network without changing too much of its structure, since for some weeks no other network could beat its performance. Yet a simple copy&paste action leads to the more complex network **Conv 9**, which is presented in Figure 5. Although the first run of this network was really promising, the loss and validation loss of the second run were worse than those of Dense 5. Although all of the tested networks have slightly different loss functions in two of the runs, the difference between the two runs of Dense 9 indicates that there seems to be some other explanation than just some stochastic effects. And indeed it turned out that the standard learning rate of Adam, namely 0.001, was not appropriate and a change to 0.0005 lead to the expected behavior.

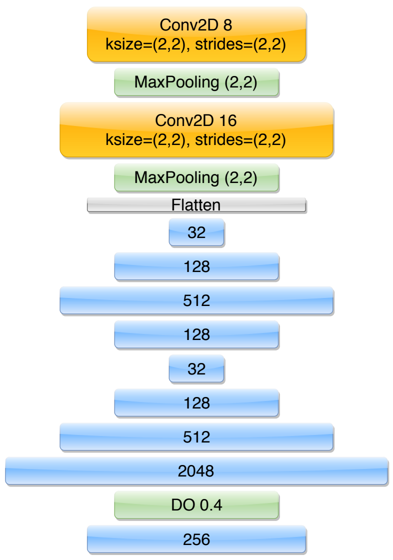
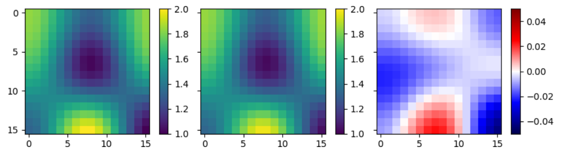


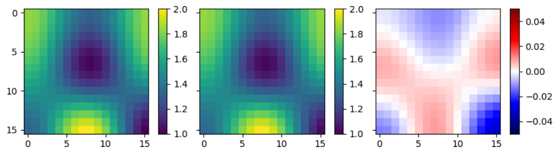
Figure 5: **Conv 9**: loss=4.3 e-4, valLoss=8.1 e-6

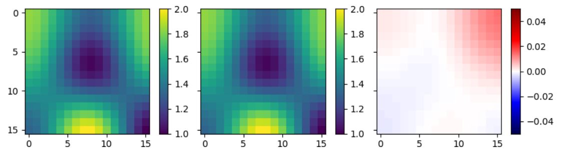
In this setting, it is also possible to narrow the second dense layer with 32 neurons to a much smaller number without getting into the situation of over-fitting. An attempt to see whether this type of net can extract the hidden information that the training data was generated with, 9 random values, resulted in Conv 11, which is more or less as accurate as as Conv 8 but much slower. Conv 10 has the same structure as Conv 9 except that the second 32-layer is narrowed to 9.

3 Discussion

The results presented in this report are gained from a total of 100000 training examples and 10000 test and 10000 validation examples packed in batches of size 2048. As with all the other structures and numbers, the batch size was determined to be the best from empirical results. To get a better impression of the networks presented above and their performance, in Figure 6 the same example is predicted by all four networks and in Figure 7 the loss and the validation loss are inserted as screenshots of Tensorboard. Since the networks have more or less the same time behavior, it can be concluded that Conv 9 is the best network presented in this report. Due to the small validation loss it can be also said that Conv 9 actually learnd the Poisson equation as described in Section 1.


(a) **Dense 2**: loss=0.0016, valLoss=1.1 e-4


(b) **Dense 2**: loss=7.9 e-4, valLoss=4.0 e-5


(c) **Conv 8**: loss=5.7 e-4, valLoss=1.8 e-5

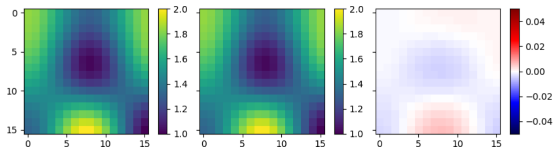
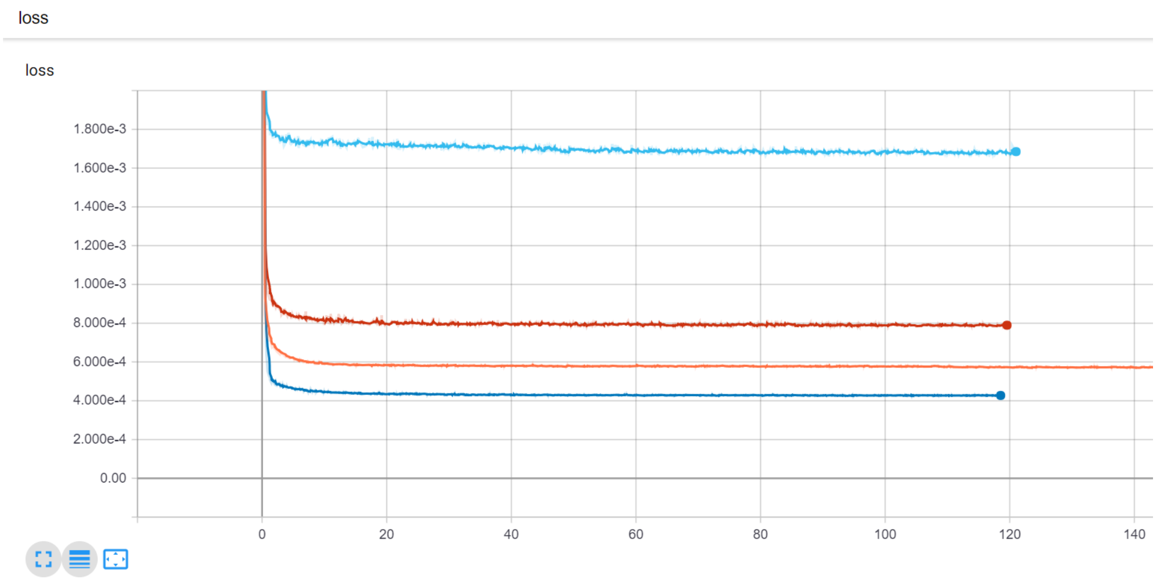

(d) **Conv 9**: loss=4.3 e-4, valLoss= 8.1 e-6

Figure 6: In all these subfigures, the first image shows the actual q, the second image the q estimated by the corresponding network, and the third image displays the difference of the actual and the estimated qs. As a rule of thumb: The whiter the third image, the better the result.

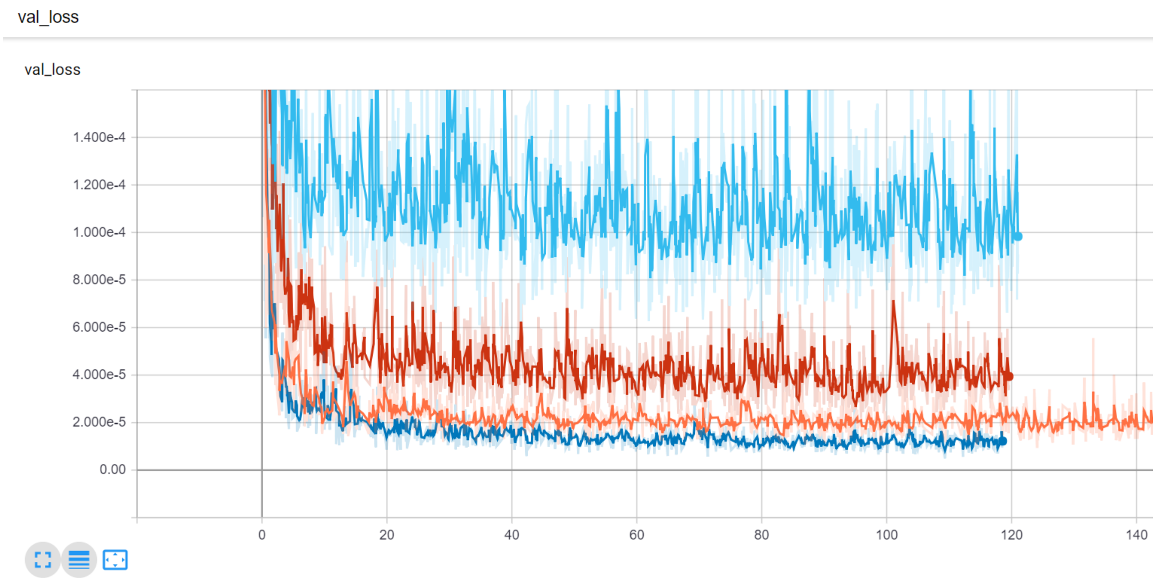
References

[GB10] Glorot, Xavier ; Bengio, Yoshua: Understanding the difficulty of training deep feedforward neural networks. In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS?10). Society for Artificial Intelligence and Statistics*, 2010

[KB14] Kingma, Diederik ; Ba, Jimmy: Adam: A Method for Stochastic Optimization. (2014)



(a) loss of all described networks



(b) validation loss of all described networks

Figure 7: Loss and validation loss of all described networks. Results of Dense 2, Dense 5, Conv 8 and Conv 9 are printed in light blue, red, orange and dark blue, respectively. While the y-axis shows the (validation) loss, the x-axis describes the training time in hours.