Flow Generation with Deconvolutional Neural Networks

Kirill Martynov, Jan Sültemeyer
Supervised by Prof. Nils Thuerey

December 22, 2017

1 Goal

The main purpose of this work is to investigate the generation of two-dimensional flow fields with neural networks. Our goal is to create qualitatively correct and physically reasonable flow velocity distributions using only geometrical information about the setup.

2 Data Generation

We consider a two-dimensional channel flow with a single circular obstacle. Thus, the setup can be encoded with three numbers – the obstacle's radius r, and the two coordinates of its center x and y.

To create the flow field for a geometrical setup defined by x, y and r values, we use *mantaflow* [4]. It is an open-source software oriented at fluid simulations for computer graphics. For a given tuple (x, y, r) we obtain the distribution of flow velocities on a uniform 64x32 grid by running a numerical simulation. An example of such a flow field – computed by *mantaflow* – is shown in Figure 1.

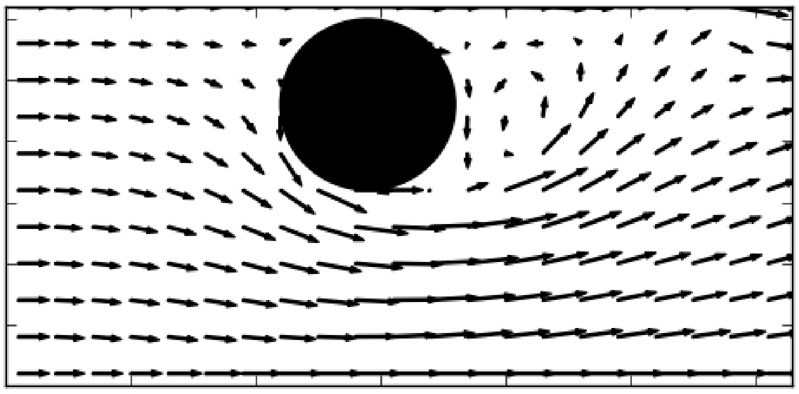

Figure 1: Example of a flow field generated by *mantaflow* (downsampled to 21x10).

A data set with 2400 samples was created by considering a range of possible values for r, x and y, and was split in the following way: 1800 training samples (75%), 300 validation samples (12.5%), and 300 test samples (12.5%). The considered tuples (x, y, r) are used as inputs to our neural network, and the corresponding outputs are the generated velocity fields. For better training, we additionally normalize the flow fields by subtracting the mean and dividing by the variance of the data set.

3 Network Architecture

Our neural network architecture is partly inspired by [3] and consists of five layers as shown in Figure 2. The three inputs of the network (x, y, r) are mapped to 4096 dimensions via linear transformations followed by $tanh()$ activation functions in two fully-connected layers. After applying

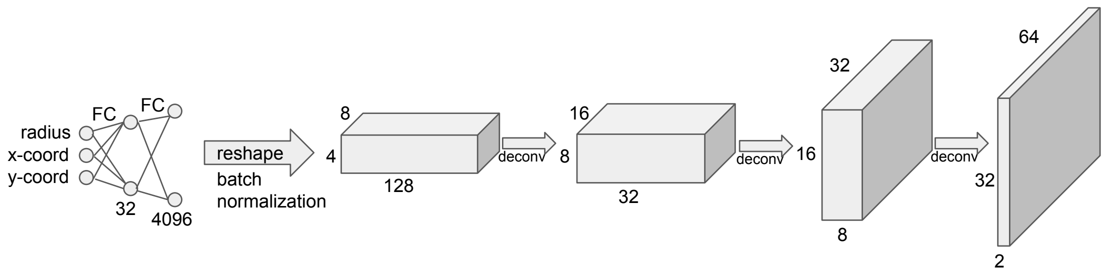

Figure 2: Network Architecture: Two fully-connected layers are followed by three deconvolutional layers.

batch-normalization [2], the output of the second layer is reshaped to a tensor of height four and width eight with 128 channels. Three deconvolutional – sometimes called transpose convolutional – layers transform it to the output shape which represents the two-dimensional flow field with a resolution of 32x64. The "volume" of the tensor is held constant in each of the layers. The deconvolutions are applied to each channel with a stride of two in both directions and a filter size of 5x5. After the first two deconvolutional layers, a leaky ReLU activation function is applied [5], and the last one uses a *tanh*() activation. The network is implemented with *tensorflow* [1].

4 Performance and Error Analysis

We use the relative L_2 error as the loss function for training, as well as a measure to evaluate the network's performance. For a velocity field generated by the network $v^{(n)}$ and corresponding target output $v^{(t)}$, this relative L_2 error is defined as

$$E = \sqrt{\sum_{\substack{1 \leq i \leq 64 \\ 1 \leq j \leq 32}} \frac{\|v_{ij}^{(n)} - v_{ij}^{(t)}\|_2^2}{\|v_{ij}^{(t)}\|_2^2}}, \tag{1}$$

where $v_{ij} \in \mathbb{R}^2$ is the velocity vector at point (i, j).

The network is trained for 100 epochs and in each of them all 1800 training samples are used. After each epoch the mean error on the training and the validation set is evaluated, and the result is plotted in Figure 3.

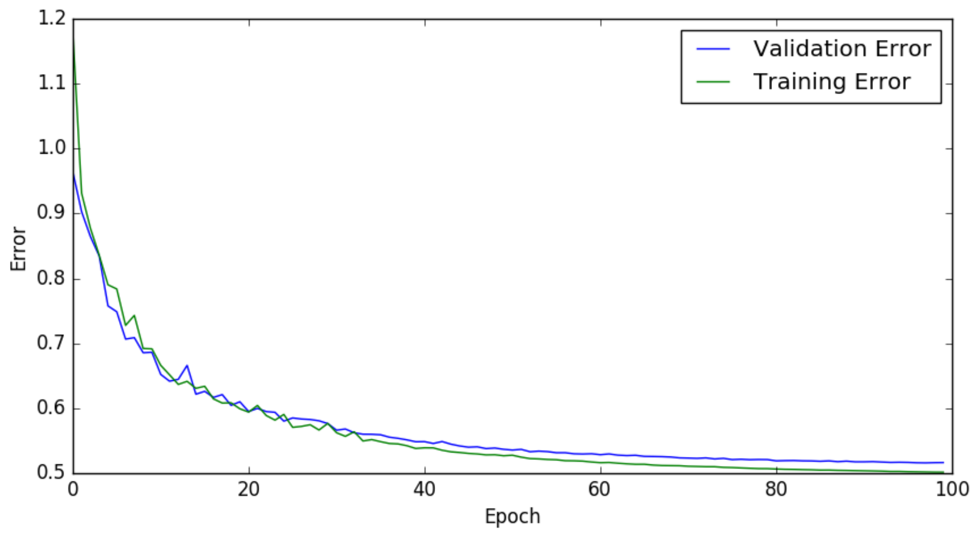

Figure 3: Training and validation error during training.

We see that the training and the validation errors are both decreasing throughout the whole training session, and have roughly the same value. This indicates that the network does not overfit the training data, because if it would, the training error would be significantly lower than the validation error. This is further supported by the fact that when we tried to train with dropout, the results did not improve.

Another observation is that the error does not drop significantly after around 60 training epochs, but is still comparably large. These two facts indicate that in order to get better results, we would have to increase the size of our network – i.e. add more parameters. This could be done by adding another fully-connected or deconvolutional layer.

A sample output of the network is shown in the top right of Figure 4. It is compared to the flow field computed with *mantaflow* for the same geometrical setup. Note that this is a sample from the test set, so the network was not trained with this particular setup. The difference of the two flow fields is plotted in the bottom row of Figure 4 – once with arrows and once color coded. We observe the maximum error directly behind the obstacle. This could be due to the fact, that the flow is the most turbulent there.

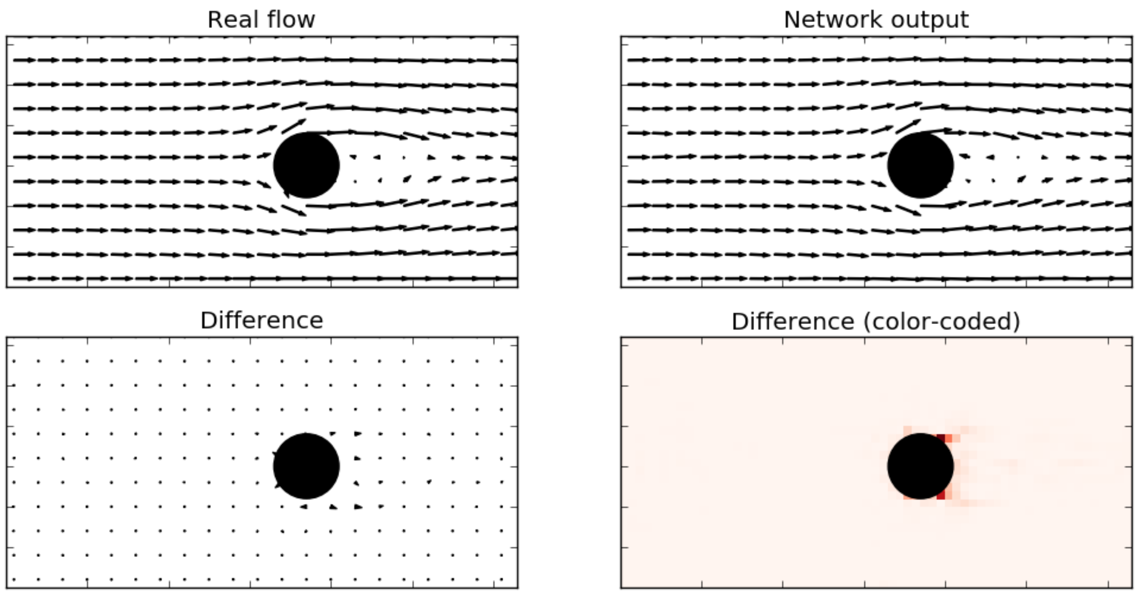

Figure 4: Results on a sample from the test set. The numerically computed flow field is compared to the output of the network for the same setup, and the difference is shown.

Although the relative L_2 error is still around 0.5 after training – cp. Figure 3 – the network output is similar to the *mantaflow* result and also looks physically reasonable. A reason for this could be that we normalize the data before training and undo the normalization afterwards for visualizing the results. The error in Figure 3 is measured on the normalized data, on the non-normalized data it is significantly lower - around 0.2.

This error is shown in Figure 5, where we analyze the error dependency on the input parameters. To produce the plots we fix one input parameter and evaluate the mean error over all corresponding test samples. The first two plots show that the mean error does not depend on the position of the obstacle. The oscillations in the graphs are due to the relatively low amount of test data points. We expect a more smooth behavior for larger test sets.

In the third plot, the dependency on the obstacle's radius is shown. Here, we see a clear positive correlation between radius and error, with only one outlier at a radius of 6. The reason for this outlier could again be the small size of the test set. An obstacle with a larger radius has a higher influence on the flow, which makes it harder for the network to predict the flow behavior.

5 Code

The code for this project can be found on *github*: https://github.com/sltmyr/FluidAnimationsDeconvNets

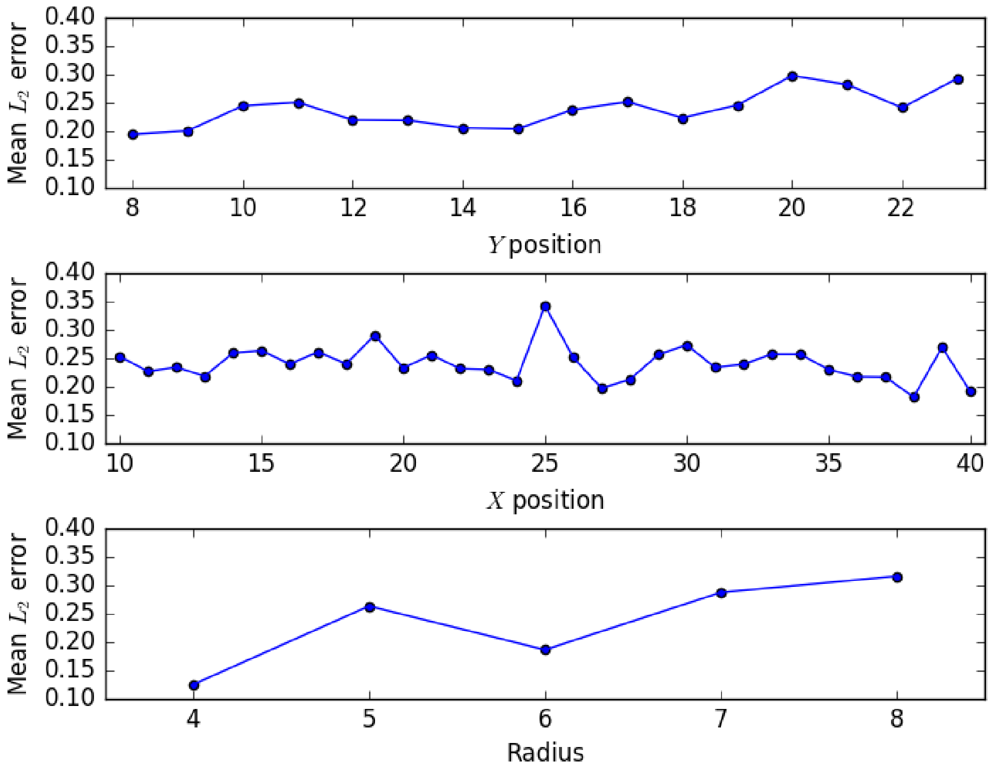

Figure 5: Error on non-normalized test data depending on the network inputs.

References

[1] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015.

[3] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[4] Nils Thuerey and Tobias Pfaff. MantaFlow, 2017. *http://mantaflow.com.*

[5] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015.