Leiden institute of Physics
Report for internship for the degree of MSc in Applied Mathematics at TUDelft

December 15, 2020

Recognizing Diffusion Behaviour with Artificial Neural Networks

Author
Daan van Velzen

Supervisor
Prof. dr. Thomas Schmidt

Abstract

Advances and proliferation in the field of *Single Molecule Microscopy* increase the demand for flexible analysis tools. The estimation of diffusion components in a trajectory and the occurrence of binding, may give important information about the underlying biology. The recent advent of advances in *Artificial Neural Networks* may provide new opportunities to develop analysis tools for microscopy data.

This article shows the applicability of ANN's to estimation of diffusion parameter. Specifically, a *Convolutional Neural Network* was trained to estimate diffusion constant, by considering the one dimensional particle location as input, and showed that reasonable estimation is possible without using physical facts about the underlying process. This achievement, though not providing any use beyond demonstration, shows that ANN models have sufficient flexibility that may be used to analyze data for features that are hard to extract by classical methods.

Experimentation with CNN's for detection of number of changes to diffusion component (switches) show no result up to this point. Experiments have been carried out to try to estimate the number of switches, to localize a switch, and to see if a one dimensional trajectory contains a switch. In this experiment, the model is trained on a sequence of displacements. None of these experiments has lead to a usable analysis tool. The large parameter space of CNN's can make experimentation and intuition difficult. Several concerns that may lead to lacking results have been proposed, but not eliminated or conclusively shown to make ANN models infeasible.

Recent advances in the field of machine learning with ANN's, show the incredible potential of the method to a wide class of problems. It is likely that further experimentation may lead to good results in the future, if current problems are resolved.

Keywords— Artificial Neural Networks, Biophysics, Diffusion

1 Preface

This report has been made as a part of an internship project as part of the MSc degree Applied Mathematics. The internship project has been provided by prof. dr. Thomas Schmidt of the Leiden Institute of Physics at Leiden University. Due to the situation regarding the Covid-19 pandemic, there was no opportunity to go to Leiden, and work was done from home. During the project contact with prof. Schmidt and the Phd-students was regular; we had group meetings twice a week. I would like to thank Thomas, Julia, Rick and Esmee for their time, the fun project and the opportunity to get a glimpse of the research that they do and the problems that they solve to do it. Moreover, I would like to thank Maria for her assistance in the research. In the past months I learned a lot and felt at home thanks to their great hospitality and help.

2 Introduction

In recent years the advent of *single molecule microscopy* (SMM) has allowed the observation of fluorescently tagged molecules in live cells, such as the glucocorticoid receptor (GR) present in the nucleus (Keizer et al., 2019) of cells in humans and other mammalians. The article by Keizer et al. (2019) provides an in depth analysis of the diffusion behaviour of the GR and identifies the dynamic states with respective diffusive components and the transition probabilities between the states. The dynamic behaviour of molecules is valuable information to biophysicists since it outlines the behaviour that a model of the underlying biological processes must be able to explain. Keizer et al. (2019) uses simulation methods on FRAP microscopy data and simulation on particle image correlations between consecutive frames to fit the occupation probabilities of the dynamic states and their respective diffusive components. The simulated diffusion behaviour is a continuous time Markov chain switching process between multiple diffusion constants, and this model, though versatile must require labor intensive fitting and careful application.

In recent years Machine Learning and Artificial Neural Networks in particular have had remarkable results in the field of image recognition, natural language processing and many other fields of application. In an article by Granik et al. (2019), an Artificial Neural Network is used to classify generated diffusion trajectories of individual particles that are either generated as a Fractional Brownian motion, Brownian motion or Continuous time random walk. Which it does with higher accuracy than classical methods. Moreover, Artificial Neural Networks outperform classical methods in de-

termining the diffusion constant of a Brownian motion when the number of consecutive frames is small.

The goal of this internship will be to investigate if Artificial Neural Networks can provide us with an additional method of analysis in multiple state diffusion like discussed in the article by Keizer et al. (2019).

2.1 Single Molecule Microscopy

Although microscopy is not the expertise of the author, some notes must be given to contextualize the importance and promise of Artificial Neural Networks (ANN's). In particular, why it is promising that ANN's outperform classical methods in determining diffusion constant when only a few frames are available. Firstly, SMM requires processing to turn the image sensor data into localization data of the individual molecules. Specialized software such as Fiji can be used to resolve the diffraction pattern of light emmitted by fluorescence to a location with nanometer scale precision. Moreover, imaging is fast, allowing for 6,25 ms between frames in the article by Keizer et al. (2019). However, the particles imaged in each frames can be fast moving, and the concentration of particles may be dense. These two problems, make tracking the individual particles difficult or impossible. There are several algorithms that allow single particle tracking, these algorithms can be imagined to roughly extrapolate the path of the particle and accept the transition of a particle to the next frame if no other particles comes close and the new location correlates with expectation. Moreover, a moving particle can of course exit the field of view and a path can end in that way.

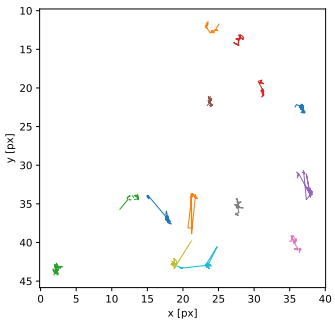


Figure 1: Single particle tracking with TrackPy(Allan et al., 2019) of 7326 frames of experimental data returns only 14 trajectories that are longer than 25 frames for 8886 particle localizations.

The idea that a diffusion trajectory can be reconstructed from microscopy

data is often an idealized view. However, since this approach is taken in the article by Granik et al. (2019), it is also the approach taken in this work.

2.2 Artificial Neural Networks

The information in this introduction section is compiled from several sources. The field of Machine Learning with ANN's (also called *Deep Learning*) was new to the author at the start of this internship. A reasonable grasp of the subject can be obtained with many excellent online resources that are available for free. The code for this report is written in python and makes use of the excellent TensorFlow (Abadi et al., 2016) machine learning platform with the Keras API (Chollet, 2015), both of which are well documented. Details on installation are provided in the appendix, code is included with the report.

In addition to the documentation of the implementation of the deep learning libraries, other sources provided insight into the theory behind ANN's. I highly recommend the MIT Deep Learning course available online (Amini and Soleimany, 2019), and the well produced material on neural networks by 3Blue1Brown (Sanderson, 2018). In addition, for more practical problems, the deep learning cookbook (Osinga, 2018) and a website called Machine Learning Mastery (Brownlee, 2020), were indispensable.[1]

2.2.1 Neurons

The base unit of ANN's is the individual Neuron, also called perceptron in some sources. An individual neuron is a function on some numerical input, with a scalar output called its activation a. The input of a neuron can be data input, such as the discplacement of a particle in our research case, or the activation of another neuron. Using the activation of one neuron as the input of another neuron in ANN's is what makes Deep Learning 'Deep'. This chaining of neuron inputs and outputs creates a network that is naturally a directed graph, where the nodes represent the neurons and the arrows represent the inputs used in a neuron.

By "learning" in Deep Learning, we mean the inference of the parameters (weights and biases) of the individual neurons. Learning is done by the gradient descent algorithm which is discussed in a later section. In order

[1] 3Blue1Brown: `https://www.3blue1brown.com/`
MIT Deep Learning Course: `http://introtodeeplearning.com/`
Machine Learning Mastery: `https://machinelearningmastery.com/`
[2] Tikz: `https://github.com/davidstutz/latex-resources`

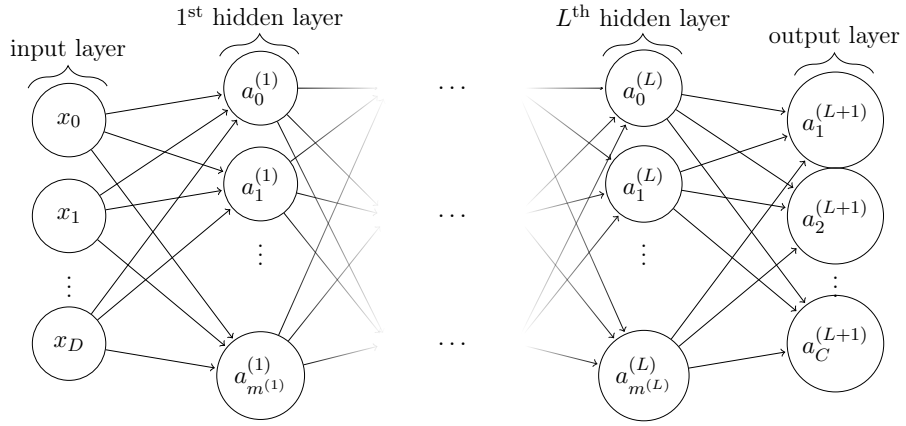


Figure 2: Network graph of a $(L+1)$-layer dense feed forward neural network with D input units and C output units. The K^{th} hidden layer contains $m^{(K)}$ hidden units. [2]

for an ANN to be *trainable* with gradient descent (to learn its parameters), we need the network structure to be acyclic. I.e., there can be no loops in the network structure. The exclusion of loops gives rise to the archetypical ANN is the so-called dense feed forward neural network fig. 2, an intuitive starting point for understanding neural networks. This network has the additional property that every path from input to output node is of the same length, which gives rise to a layered structure. Other network structures are possible, and the requirement of acyclic structure can be broken in some cases. We will discuss network structure (architectures) in a following section. The dense feedforward neural network depicted in fig. 2 takes in input of dimension D and is trained to give a C dimensional response. Typically responses can be either classification probabilities or any other relevant numerical output, such as diffusion constant in our case. The layers of neurons that are neither input nor output layers are called hidden layers. In order to train the network, we must have access to labeled training data, data for which the correct output response is known.

We now have an idea of the role of individual neurons, but the exact specifications are not yet clear. Neurons are functions with a multidimensional input and a single numerical output called activation. The activation of neuron i in the K'th layer of a dense feed-forward ANN is

$$a_i^{(K)} = \phi\left(w_{i,0}^{(K)} a_0^{(K-1)} + w_{i,1}^{(K)} a_1^{(K-1)} + \ldots + w_{i,m^{(K-1)}}^{(K)} a_{m^{(K-1)}}^{(K-1)} + b_i^{(K)}\right). \tag{1}$$

Here $\phi : \mathbb{R} \to \mathbb{R}$ is called the activation function which will be discussed in a later section. The parameters $w_{i,j}^{(K)}$ are called the weights, and they

correspond to the arrows of the directed graph. Intuitively, the magnitude of the weight between two neurons indicate how strong the effect of one neuron on the next is. The parameters $b_i^{(K)}$ are called the biases. The notation of eq. (1) is cumbersome and looks cleaner in vector notation:

$$a^{(K)} = \boldsymbol{\phi}\left(W^{(k)}a^{(K-1)} + b^{(K)}\right).$$

Here we let $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^n$ be defined as

$$\boldsymbol{\phi} : \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} \mapsto \begin{pmatrix} \phi(x_0) \\ \phi(x_1) \\ \vdots \\ \phi(x_n) \end{pmatrix}.$$

Note that if we let go of the layered structure of the dense feed forward neural network, we will have to modify the notation. However, the principle of the neuron is aptly described in this context and is easily extendable to other use cases.

2.2.2 Activation Functions

In the Keras and TensorFlow workflow it is common to construct an ANN out of layers, such as the dense layers discussed in the previous section. Normally the neurons in a single layer will all have the same activation function. This is the default in the Keras layers api. A foundational theorem that demonstrates the promise of neural networks is the *Universal Approximation Theorem* of Cybenko, Hornik, Leshno and Pinkus. The refined version proves that a single hidden layer model in which the hidden layer has arbitrary size, can represent any function to arbitrary precision, as long as the activation functions are non-polynomial. The study of the theoretical underpinnings of ANN models is beyond the scope of the report, but note that the activation function must be chosen to be non-polynomial (although in practice, non-linearity may be enough).

In choosing an activation function for a layer, there are several other considerations we must take into account (Jadon, 2018). Firstly, for an output layer, it is important to make sure that the range of the activation function is suited to the range of the required output. Moreover, activation functions must be differentiable a.e. to allow for backpropagation. Another consideration is saturation. Where an activation function reaches a large flat region, a change in the input no longer leads to meaningful change the in activation of the neuron. This problem is aggravated when there are multiple layers with a saturating activation function.

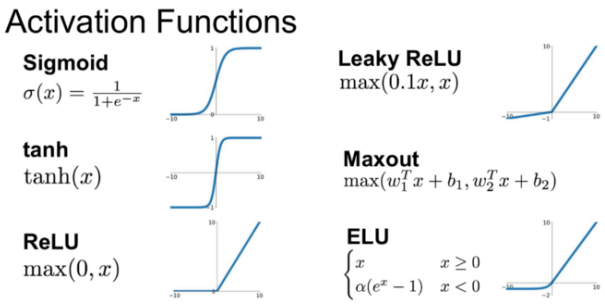


Figure 3: Overview of commonly used activation functions. From article (Jadon, 2018)

The activation functions in fig. 3 are amongst the most common activation functions. The sigmoid and tanh functions are activations that have been considered in seminal ANN literature. In recent years, in the field of image classification, the ReLU or Rectified Linear Unit has been very successful. One problem it causes, is that there is no gradient for negative input, which means that backpropagation can not train its weights when the input is negative. This is called the dying ReLU problem, which can be solved using the Leaky ReLU. The maxout is another option to solve the problem, but it doubles the required parameters. The ELU or Exponential Linear Unit is continuously differentiable, which is a desirable property. In addition to these activations, for output layers of classification models, the *SoftMax* function has good properties. It is a continuously differentiable approximation of the arg max function, which additionally is normalized so that the output activations add up to one. In vector notation:

$$\operatorname{softmax}_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}.$$

2.2.3 Loss Functions and Gradient Descent

In order to learn the model parameters of the ANN, we want to find the parameters that make the model match the output to the labels of the training set as close as possible. To specify how well the ANN is working, a Loss function is specified. Specifying a problem in terms of optimizing parameters to reduce a loss function is a basic technique from mathematical optimization, which is widely used in statistical estimation, decision theory, hypothesis testing and regression. In general terms, we wish to minimize our expected loss with respect to some probability measure on the population space. In general, we do not have the ability to minimize the expected loss on the entire population and/or we do not know the

distribution of the population. In most practical applications of statistics we therefore look at a sample from the population, which introduces the danger of overfitting. More on this in the section on callbacks. In this report, most ANN's are fitted to generated data, which clears the risk of overfitting, but may still have over- or underrepresented subsets of population space if the probability measure is chosen incorrectly.

Training the model is typically done using gradient descent. The idea of gradient descent is that it is impossible or computationally infeasible to find the parameters that find the global minimum of the loss function for the entire sample set. As discussed before the global minimum would likely result in overfitting. Instead, we update the parameters iteratively. For the first iteration, weights are chosen by an initializer (at random or optimizing it for fast learning). For each consecutive step the gradient of the loss with respect to the parameters are calculated for a subset of the training set. These subset are called batches, and add some stochasticity to the gradient descent. Calculating the gradient requires that the activation is differentiable, which explains this requirement. Weights are then updated against the gradient, to lower the loss function. This update step updates with a small step size, to prevent overshooting the target. This step size (learning rate) is controlled, and decreased with each iteration by an optimizer such as ADAM. The most noteworthy way of determining the partial derivatives of the loss function with respect to each parameter is backpropagation. Backpropagation algorithms use automatic differentiation techniques to calculate the partial derivatives of the loss with respect to each parameter. Since the total loss as a function of the parameters is a function composited of linear steps and nonlinear activation functions, the chain rule is used repetitively. Because we are using the chain rule, on a composite function, we can work from outside in efficiently. This corresponds to working from output to input, hence giving rise to the name backpropagation. Working back from layer K to $K-1$ allows the intermediate partial derivatives needed for layer (K) to be employed for the previous layer, thus making the process memory efficient. The workings of gradient descent and backpropagation are highly abstracted in the Keras and TensorFlow API.

It is finally important to note that the choice of loss function determines in a large amount whether the model will train or not.

2.2.4 Architectures

So far we have focused our attention on the dense feed-forward neural network. In this model, the connections between layers form complete bipartite graphs; every neuron in the $K-1$'th layer is connected to every

neuron in the K'th layer. This model is flexible but has a very large parameter space which makes training slow. In the article by (Granik et al., 2019), convolutional neural networks are extensively used. A convolution of spatial or temporal data only has connections to a collection of neurons that form a neighbourhood as shown in fig. 4. Moreover, for each of these neighbourhoods, the weights and biases to neurons in the subsequent layer are the same. Trained models with this architecture tend to encode the presence of certain features in the data. This encoding is unsupervised and may improve on the human ability to simplify input space. Moreover, consecutive convolutional layers may encode a hierarchy of more complex features. This property has made convolutional layers popular for image classification. Here a neighbourhood of the input space represents a subregion of the analyzed image. Convolutional layer weights are often called filters in this context, as the trained weights can act as edge-detection filters etc.

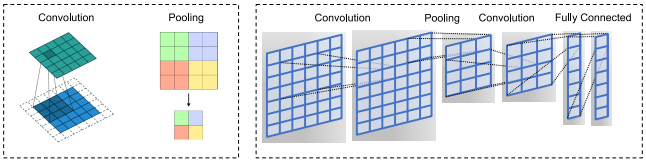


Figure 4: Graph of convolution as application of a filter to a 2d neighbourhood on the input. Also shown is pooling, or downsampling. From article (Maier et al., 2019)

A different type of architecture that is not employed in this report is the Recurrent Neural Network. These networks break the requirement that architectures must be acyclic. Recurrent Neural Networks (RNN's) are applicable to arbitrary length and temporal input and may warrant further investigation in the context of this reports goal.

Finally we note, that ANN's are typically not exclusively made up of neuronal layers. Several other layers are commonly used such as pooling, normalizing, reshaping and regularizing layers. These layers tend to be included to improve training, make more complex architecture possible or make the model more parsimonious.

2.2.5 Callbacks

Callbacks are functionalities that can be performed before or right after a training batch, or number of training batches (often called epoch). They provide high level functionality that is practical to use in training the model. Returning to the topic of overfitting. Training a model on a sample of a population introduces the risk of overfitting. Overfitting

occurs when the performance of a model in-sample continues to improve, but its out of sample performance deteriorates. A useful callback is the early stopping callback, which halts training when the performance on a keep-out validation set no longer improves. This model works nicely when combined with the checkpointing callback, which allows you to save the model that has the best performance of all training cycles. In general it is advisable to include cross-validation in any machine learning model that is trained on a non-exhaustive sample.

3 Methods and Results

Given the relative inexperience of both the author and supervisor to modeling with ANN's, we take an incremental approach to the goal of recognizing multiple state diffusion. The general expression for the diffusion constant D is

$$\sigma^2 = \langle x^2 \rangle = 2Dt,$$

where x denotes the displacement. In this report, we will try to estimate σ as it is a practical quantity to use in the generation of trajectories.

3.1 Distinguishing Fast and Slow Diffusion

The first approach we will discuss is training a model to estimate σ based on the trajectories of single particles. The methods used in this first model leave room for improvement and a model with better performance will be discussed later.

We start out by simulating 20.000 trajectories of 100 step size. The trajectories are generated by generating the the starting point $U \sim \text{Unif}(0, 50)$ and 99 increments $X_i \sim \text{N}(0, 1)$, such that $\sigma X_i \sim \text{N}(0, \sigma^2)$. We then take the cumulative sum along the time dimension.

This dataset is then split in a training set, validation set and test set. The training set is used in gradient descent. The validation set is left out and used to cross validate the model. The test set is left out to the very end and used to calculate performance metrics for the model.

The model used is derived from the work of Granik et al. (2019) and show in fig. 5. The model consists of 4 convolutional filters, the results of which is aggregated by max-pooling, which down-samples the input, making the model more sparse.

The model uses ReLU activations and will be trained for 20 epochs on a batch size of 300. The model is trained with mean squared error loss,

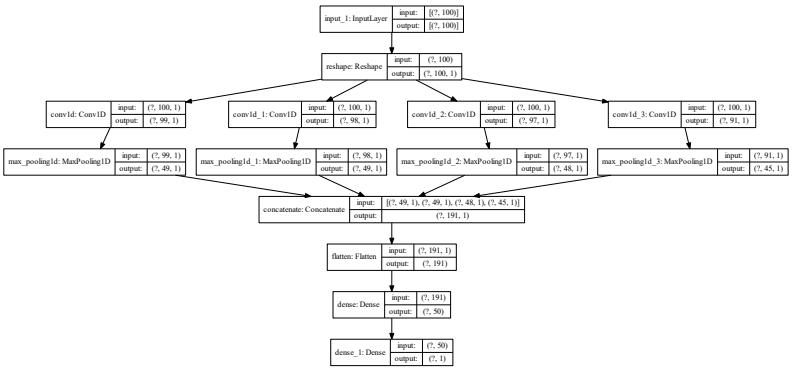
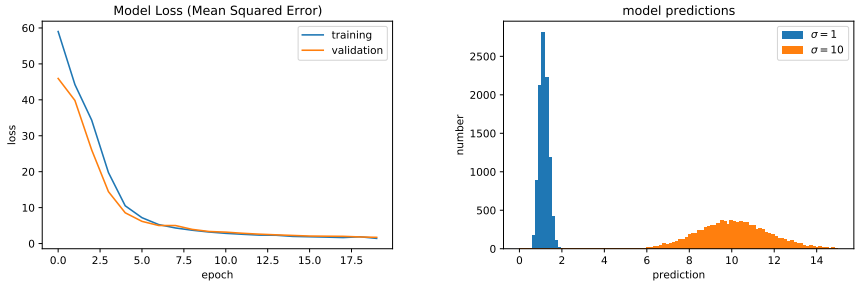


Figure 5: Architecture of convolutional ANN used to discern between fast and slow diffusion.

which penalizes large deviations from the estimate harshly. Moreover, it picks up on both bias and estimator variance and is therefore a popular choice.

Figure 6



(a) Learning curve for distinguishing between fast and slow diffusion.

(b) Predicted value of σ for 20.000 randomly generated trajectories with $\sigma = 1$ for 10.000 and $\sigma = 10$ for the other 10.000.

The model in intended to work on the particle locations and not on the increments, which are independent. The model has filter sizes of 2, 3, 4 and 10. It is expected that a functioning model trains the size two filter to be the difference of subsequent locations, so that the model has the independent increments and can determine the variance of them. The result of training is shown in fig. 6. We observe that the training seems successful, the model MSE (fig. 6a) levels of at a value of around 2, and the histogram of predicted diffusion constants (fig. 6b) correctly discerns between fast and slow diffusion. The filter for the convolutional layer of

size two is:

```
[-0.30045655, 0.4582819 ]
```

The next step is to make some observations about the trained model. First we see that the convolutional layer of size two does indeed seem to encode for the difference of subsequent points in the trajectory, but does not assign the same importance to the left and right point. Moreover the training loss seems to plateau at a value of 2, but the number of training cycles has been chosen arbitrarily. This means that it is not yet possible to say if the model has been trained sufficiently. Moreover, we are training this model from data that is generated at random. This means that as long as the distribution we sample our data from is representative of the use case, there is no risk of overfitting if we generate data on the fly. Finally, for this first trained model, we have limited the training set to only $\sigma \in \{1, 10\}$. This is an oversimplification of the intended use case in which we would not know the value of σ, which could be in several orders of magnitude.

3.2 Estimating Diffusion Constant

We now incorporate the observations from the previous model to make a model that estimates σ. In contrast to the earlier model we now generate training data on the fly, which is more memory efficient and removes the risk of overfitting. Values of σ are drawn from a Weibull distribution with scale parameter .7. This distribution has a heavy tail and therefore has outliers that can be several orders of magnitude larger than the average. This distribution is chosen because we want consistent results over a range of magnitudes. As a loss function we use the mean absolute error also called mean absolute percentage error (MAPE) in Keras. We choose this loss function, since it consistently penalizes the size of the relative error across a range of magnitudes, so that for uniformly distributed samples, the relative absolute error should be constant across the domain.

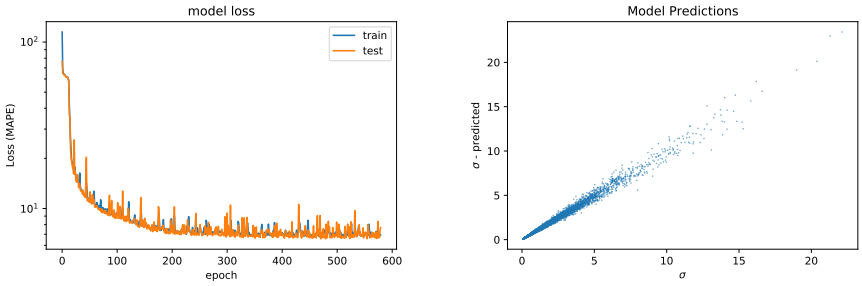
In training the improved model, we make use of the *early stopping* callback. This callback validates the performance of the model with generated validation data, and only stops the training when no improvement in the validation loss occurs for 40 epochs. Moreover we make use of the *model checkpoint* callback, which saves the model with the lowest overall validation loss.

The architecture used for this model is the same as in fig. 5. Since the distribution we sample from is likely to contain outliers of a different magnitude, we train this model for 30 steps with a batchsize of 1000, per

epoch. In this way the training gradient is fairly consistent in the training process. The training output shown in fig. 7 shows that training shows marked improvement up until the 200th epoch. After this the validation loss remains around 7 percent. The checkpointed most optimal model finally reaches an MAPE of $6,6202$ after testing with 10.000 generated trajectories. Meaning that we trained this model to have an expected relative error for σ of around 6,6 percent. We compare this to the performance of $\hat{\sigma} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \langle x \rangle)^2}$, where x denotes displacement. The expected relative error we approximate by bootstrapping. In this manner we find that the MLE estimate for σ slightly outperforms the ANN, as it has a value of $5,8329$ percent after verifying on the same 10.000 generated trajectories.

Moreover, for $\sigma = 1$ we can consider $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \langle x \rangle)^2$. $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \langle x \rangle)^2 \sim \frac{\sigma^2}{n}\chi^2_{(n-1)}$. This means that $\mathrm{Var}(\hat{\sigma}^2) = \frac{2\sigma^4(n-1)}{n^2}$. Meaning that in this case the variance of estimates for $\sigma = 1$ would theoretically be 0.0200. If we approximate the distribution of $\hat{\sigma}^2$ as a normal, and take the estimate to be unbiased (which is reasonable), then mean relative error for σ^2 (mean of folded normal distribution) would be 0.01600. This value is not directly comparable to the current performance metric. Training the model for regression on σ^2 instead of σ does not achieve results comparable to fig. 7a as the learning curve for this model fluctuates irrespective of chosen distribution of σ and loss function. Therefore it appears that the problem must be stated in terms of σ or it appears to be ill-defined and does not train. Here we run into a practical considera-

Figure 7



(a) Learning curve for estimation of σ. MAPE loss on log-scale.

(b) Predicted value of σ for 10.000 randomly generated trajectories with $\sigma \sim \mathrm{Weibull}(.7, 1)$.

tion in model selection. Modeling problems with ANN's requires that the architecture, modeled quantity, activation functions, samples, initializations etc. together form a combination with a smooth and wide gradient. Troubleshooting the training performance requires experience and good

grasp of underlying principles. Apart from simplifying the model or the problem, the recourse to take when a model does not train is limited.

3.3 Discerning Switching Behaviour in Diffusion Trajectories

The approach in this section is largely based on the article by Granik et al. (2019). In the paper an ANN is trained to discern between three types of diffusion; Brownian motion, fractional Brownian motion and the continuous time random walk. Aditionnaly, noise is added to the output signal, which does not cause For the purpose of aiding analysis of experimental microscopy data, an easy way of recognizing diffusion with multiple diffusion components would be a good aid for analysis such as it is done in the article by Keizer et al. (2019). For the purpose of this report, we will call a change in the diffusion component a switch. Our method for this problem will be to see if the model proposed in the article by Granik et al. (2019) can be adapted to the problem at hand, given its good performance in the article. In addition the problem in the original article may be considered more difficult, since the difference between the various diffusion behaviours is small. Fractional Brownian motion is very similar to Brownian motion, but the increments at several values of lag are not independent. For continuous random walks, a Poisson point process determines the times at which the position of a particle changes. These displacements are usually independently distributed normal random variables. For the purpose of this report we will only try to discern Brownian motion and Brownian motions with switching.

3.3.1 Data Generation

In concordance with the article by Granik, we look at only one spatial dimension, reducing the diffusion trajectory in 2d to a 1d sequence. In order to hopefully reach the possibility of integration with the current software by Granik et al. (2019)[3], the data generation will be written to interface with the higher level code of the repository. The switching process is modelled as Poisson point process, where switches in the diffusion component occur at the random time points. At each of these switching times, the model picks one of the other diffusion components at random from a list, making sure that there is an actual switch taking place. This means no consecutive equal picks can be made. The fact that the switching events take place following a Poisson Point process, is a change from the original

[3]Github Repo (Granik et al., 2019): `https://github.com/AnomDiffDB/DB`

article by Keizer et al. (2019), in which the Markov chain model has different transition probabilities for the various states of the particle. However, it should not make a difference in the training of the model, since the diffusion changes are sufficiently noticeable. The resulting trajectories can be seen in fig. 8.

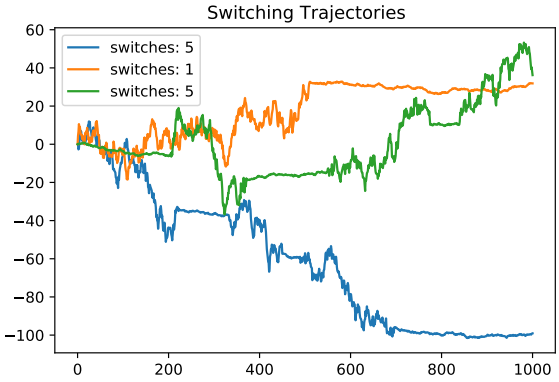


Figure 8: Three trajectories of particles with switching behaviour. $\sigma \in \{1, 10\}$

In accordance with the work and code by Granik et al. (2019) a preprocessing step is added to the trajectories, which means that the model is trained on the displacements of the trajectory, not on the positions of the particle.

3.3.2 Subquestions

The initial goal for this section was to directly classify the trajectory as either switching diffusion component, or a simple brownian motion. Due to difficulty training the ANN for the most general problem, repeated steps have been taken to simplify the problem. In this process the problem has been divided up into several progressively easier subproblems:

How many switches are there? In this problem we allow switches to occur randomly and want to know how many switches took place. This problem can either be seen as a classification or as an estimation problem. However, using loss functions such as the mean squared error and mean absolute percentage error for output which should be integer valued is problematic.

At what time does a switch occur? For this problem we simulate trajectories which all have a single switch, occurring at a random time. We train the model to correctly determine the moment at which the switch occurs.

Is there a switch? The simplest question we try to answer is to see if there is a switch. In this problem we generate trajectories with either a single switch or no switch at a fixed point in time. The problem is modeled as a classification problem.

3.3.3 Models

For the problem of discerning switching behaviour, two models and many variations thereof have been considered.

Model 1 The first model considered is the model that proved successful in section 3.2. The model contains a single convolutional layer which may do the job of locally estimating diffusion constant. Three fully connected layers that come after may be able to discern switching, but additional depth may be necessary.

Model 2 The second model is directly taken from Granik et al. (2019). The model has three layers of convolution and normalization end concludes with max pooling. Each convolution size has 32 different filters that are applied to the trajectories. The model is originally used for the classification of type of diffusion (Brownian, Fractional-Brownian and Continuous time random walk).

For both model a wide number of variations has been considered. First it is important to note that for the subproblems of our larger project, the output can either be categorical or numerical in nature. This implies the choice of several types of loss functions and categorizations of the generated trajectories. In the architectural sense, the advice of Maria later in the project was to build models incrementally, to add models only if a simpler model does not work, or try to find a minimal working example and expand on it by adding layers. This advice came at a later stage in the project, and was counter to my actions up to that point.

As tunable parameters for the training. I considered learning rate, batch size of training and validation, training epochs (usually controlled by earlystopping callback), learning rate, loss function, activation functions, type of optimizer and variations in initialization of model weights and

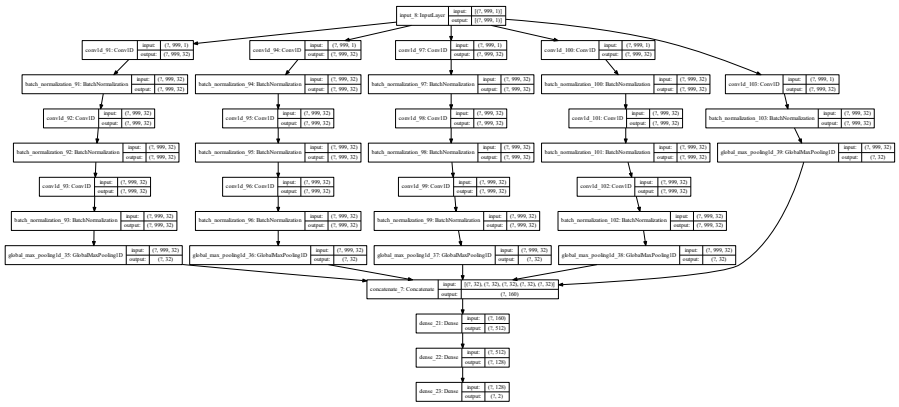


Figure 9: Classification model as used in the article by Granik et al. (2019). Three layers of convolution and normalization layer and max-pooling, with one shortcut. two hidden dense layers.

architecture. In particular, for the second model the exclusion of the normalization layer has been extensively tried due to worries about the features in the training data disappearing under normalization. More on this will be discussed in later sections.

The strategy at the start of the project was to find a model of sufficient flexibility which was compliant with my computing resources and to see if there were tunable parameters which could allow the model to fit to a good minimum. This approach demonstrates my inexperience with CNN's as a more structural bottom up approach is likely to have been more productive.

3.4 Recognizing Diffusion Behaviour

The following subsections show some limited results of the experimentation, in which none of the models provide a good fit for the training data. For all three of the subquestions results show estimators that are qualitatively unsound, as they tend to assign estimates at random around an average that best suits the sample population. For this reason, quantitative analysis of the different models has not been undertaken, since the model is qualitatively wrong. This section will primarily focus on the reasons that have been considered for the bad performance, and the motivation of the simplifications of the main question that have been attempted.

3.4.1 How many switches are there?

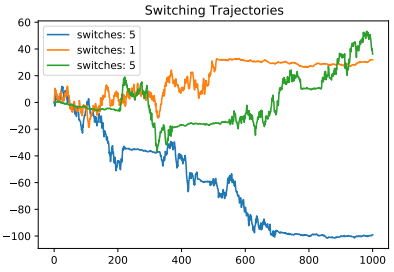
In this problem we allow switches to occur randomly and want to know how many switches took place. This problem can either be seen as a classification or as an estimation problem. In this experiment, switches are generated by poisson point process with an average occurence of once every 100 steps. Trajectories of 1000 steps are generated and switches are between values of $\sigma \in \{1, 10\}$. The number of switches is stored as a label to be estimated.

Using loss functions such as the mean squared error and mean absolute percentage error for output which should be integer valued is problematic. A possible solution to this is to consider Poisson loss as an option, which has been used. Other options are MAPE (which penalizes relative to number of switches, which may be considered lacking in ambition) and MAE.

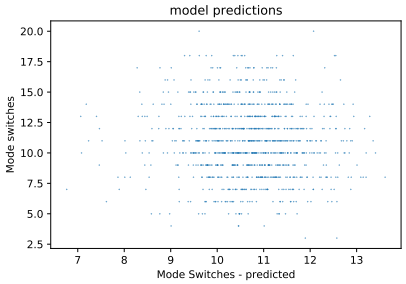
In many iterative attempt to tune the parameters of the models to allow a fit, no good estimator has been obtained. The original reasoning behind the choice of model is that a single layer of convolution allows to obtain an estimate of diffusion constant to some precision. Consecutive layers of convolution, should therefore be able to detect the boundaries between regions with different diffusion constants. A third layer may then be used to obtain higher level information about for instance, number of switches. My idea of optimism about the flexibility of the model, is not to supervise the model learning to conform to this structure, but allow the learning process to come to a minimum even if it does so in a less intuitive way.

In fig. 10, the training process for the prediction of number of switches is shown. It is apparent that this estimate in itself is no good. Several reasons for the lack of a result have been proposed.

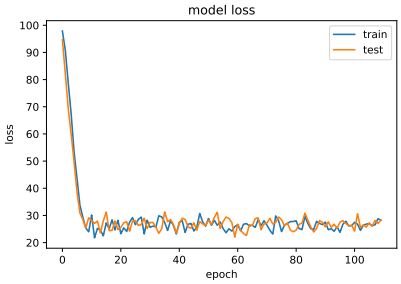
- The original work by Granik et al. (2019) considers diffusion behaviours that are characterized by dependence between timesteps. Fractional brownian motion is brownian motion with non-zero autocovariance between steps of lag. Convolutional neural networks are good at discerning local patterns, even if they are small. The diffusion steps in our project are all centered around zero, the average displacement shows dependence, not the displacement steps themselves. For a convolutional filter, the input of the activation function is therefore centered around zero. It is only the variance of the distribution that changes and convolutional neural networks have no squares of input as input. The proposed solution for this problem is to train the model on the magnitudes of x^2 instead of x. This idea goes against my original plan of doing minimal preprocessing, and trusting the model. Results for this attempt were no



(a) Generated trajectory with varying number of switches



(b) Predicted number of switches for 1000 generated trajectories



(c) Learning curve for estimation of number of switches.

Figure 10: Training process of a typical model on the estimation of number of switches.

better than before. This is one of the main difficulties of working with CNN. There are no intermediate markers for succes, because the parameter space is large and obscure, as it bears no relation to intuition or physical quantities.

– The nature of convolutional filters is that they are a linear transformation applied to a local region, to which a nonlinear activation function is applied. The local nature may make it dificcult to see distinct change in diffusion constant, as the constant is better estimated along a larger region. However, the advice from Maria, which I have taken as a good rule of thumb, is that any data feature which is discernable by eye, should be discernable by CNN. A good method, could be to use strides in the filter, and use pooling to make the distinction between diffusion constants sharper. This has however not lead to marked improvement in results.

– Lastly, the normalization layer, though intended to allow input from a wider number of degrees of scale, and to prevent saturation to the linear regions of the activation functions, may inadvertently remove

features from the data. Testing and study of the documentation has been performed and leads to conclude that this is not the case. Normalization constants are computed during training on all training samples. Since this is a preliminary, proof of concept phase, the performance is measured on trajectories sampled from the same distribution, so normalization applied to the trajectories will only rescale. It is more difficult to grasp the effect of normalizations in between convolution steps, which is why the model from Granik et al. (2019) has been applied and tested without the normalization layers as well as in the original manner.

After experimenting and considering the flaws of this approach, a strategy of subsequently simpler subquestions was adopted in order to gain results and have a baseline for comparison.

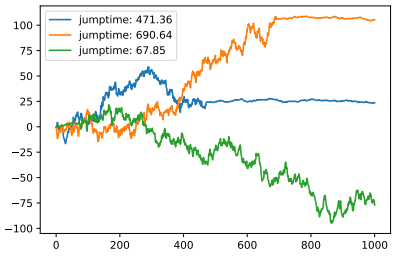
3.4.2 At what time does a switch occur?

For this problem we simulate trajectories which all have a single switch, occurring at a random time. We train the model to correctly determine the moment at which the switch occurs. The loss functions used are MAE and MSE.

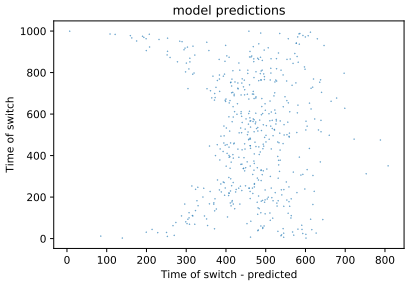
Again no satisfactory estimator is obtained after prolonged experimentation. However, the estimator does show interesting behaviour fig. 11 and though there is no great covariance between predictions and actual times of switching. The model obtained does not predict a switch before $t = 500$ if the switch takes place before $t = 400$ or after $t = 600$. This behaviour is unexpected, and though the performance is still not greater than than predicting the population average for every input, it does show that the model manages to discern some properties in the input. These properties are not correctly mapped by training. Another way of putting it, is that the estimator does not predict a switch around the beginning or end of the trajectory if there is a switch in the middle of the trajectory. The mechanics that lead to this behaviour are not well understood.

Again it is my suspicion, that the independence of increments of the displacements thwarts feature detection, as it is antagonistic to the strengths of CNN's.

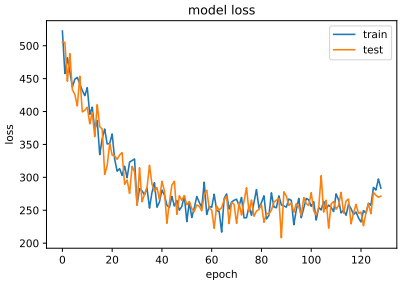
Positive about this iteration of experimentation is that the predicted time of switch does bear a relation to features in the data. It is therefore logical to ask why the learning of the CNN does not yield a better result. In the end, there must be some gradient that leads the CNN to find a feature in order for it to affect the prediction. The author considers it likely that the highly stochastic and largely independent nature of the input data,



(a) Generated trajectory with one random switch



(b) Prediction of time of switch



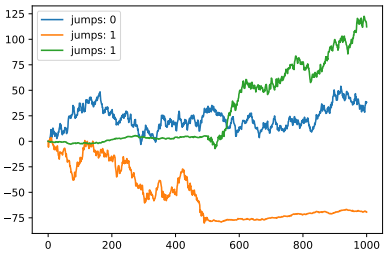
(c) Learning curve for estimation of switch time. Mean average error = 260.2427

Figure 11

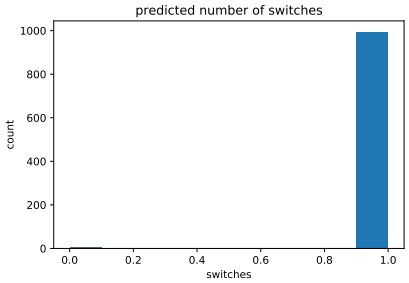
makes the effect of training steps too volatile and thus prevents gradient descent. A solution to this would be to drastically enlarge the training batch size, to thus lessen the effect of stochasticity. It would be a good idea to attempt training the model on a computer with greater computational resources. To investigate the feasibility of this, access to the TUDelft High Performance Computing cluster has been requested (although in an earlier stage of research). Operation of this system did appear to involve considerable investment of time and since gaining assistance was difficult, this course has been abandoned.

3.4.3 Is there a switch?

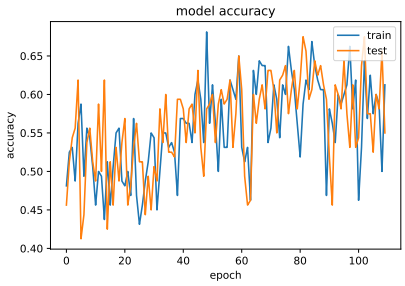
The simplest question we try to answer is to see if there is a switch. In this problem we generate trajectories with either a single switch or no switch at a fixed point in time. The problem is modeled as a classification problem. Either there is a switch or there is no switch. In light of the earlier disappointing results, the switch always occurs at $t = 500$. This is done in order to provide an easy baseline for experimentation. The results



(a) Generated trajectory with switch at fixed position



(b) Predicting whether trajectory switched for 1000 trajectories., accuracy = 0.5035



(c) Learning curve for assessing switch occurrence.

Figure 12

of training are shown in fig. 12. Training seems not to improve the value of the loss function and the categorical predictions for the loss function place all estimates in one category. The reason for this eludes the author, it seems that this problem should be easy enough for a CNN to work, even given the earlier indicated potential difficulties.

It is noteworthy, that should it be possible to train a CNN to recognize a switch for any time t, that this also allows the CNN to be used in localizing the switch. The CNN architecture has convolutional filters, that even if layered, correspond to a specific region (a number of time steps) in the sample trajectories. This means, that should the CNN lead to an accurate model, the final convolutional layer activations give an indication of where the model deems the switch the most likely to have occured. This means that a classification CNN model can be leveraged to lead to a detection model. This adaptation of CNN's is common and could possibly be used in future experimentation.

4 Conclusion

Advances and proliferation in the field of *Single Molecule Microscopy* increases the demand for flexible analysis tools. The estimation of diffusion components in a trajectory and the occurrence of binding, may give important information about the underlying biology. The recent advent of advances in *Artificial Neural Networks* may provide new opportunities to develop analysis tools for the analysis of microscopy data.

This article shows the applicability of ANN's to estimation of diffusion parameter. Specifically, a *Convolutional Neural Network* was trained to estimate diffusion constant, by considering the one dimensional particle location as input, and showed that reasonable estimation is possible without using physical facts about the underlying process. This achievement, though not providing any use beyond demonstration, shows that ANN models have sufficient flexibility that may be used to analyze data for features that are hard to extract by classical methods.

Experimentation with CNN's for detection of number of changes to diffusion component (switches) show no result up to this point. Experiments have been carried out to try to estimate the number of switches, to localize a switch, and to see if a one dimensional trajectory contains a switch. In this experiment, the model is trained on a sequence of displacements. None of these experiments has lead to a usable analysis tool. The large parameter space of CNN's can make experimentation and intuition difficult. Several concerns that may lead to lacking results have been proposed, but not eliminated or conclusively shown to make ANN models infeasible.

Convolutional methods may not be ideally suited to distinguish non-local features. Switches become apparent when assessing the diffusion component of the particle at multiple distinct times. The architecture of multiple convolutional layers makes the model inherently more suited to local features in the data. A wider choice of architectures may need to be considered.

Linear nature of convolutional filters is not ideally suited to establish an estimate of variance from the distribution of displacements. This may cause a gradient descent algorithm to fail, because the gradient is not wide and the stochastic nature of the input may cause erratic training. A possible solution may be to train the model with far larger training batches. This may eliminate erratic training behaviour.

Large number of training parameters makes purposeful improvement difficult. The experimentation process has involved extensive testing of a large number of different parameters to lead to better results. Training an ANN model requires experience and may be considered an art. Further

experimentation by a researcher who is more familiar with this family of models, may yiel better results.

Recent advances in the field of machine learning with ANN's, show the incredible potential of the method to a wide class of problems. It is likely that further experimentation may lead to good results in the future, if current problems are resolved.

References

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*, available at `1603.04467`.

Allan, Dan, Casper van der Wel, Nathan Keim, Thomas A Caswell, Devin Wieker, Ruben Verweij, Chaz Reid, Thierry, Lars Grueter, Kieran Ramos, and Others. 2019. *soft-matter/trackpy: Trackpy v0.4.2*.

Amini, Alexander and Ava Soleimany. 2019. *MIT 6.S191: Introduction to Deep Learning*.

Brownlee, Jason. 2020. *Machine Learning Mastery*.

Chollet, François. 2015. *Keras*.

Granik, Naor, Lucien E. Weiss, E. Nehme, Maayan Levin, Michael Chein, Eran Perlson, Yael Roichman, and Yoav Shechtman. 2019. *Single-Particle Diffusion Characterization by Deep Learning*, Biophysical Journal **117**, no. 2, 185–192.

Jadon, Shruti. 2018. *Introduction to different activation functions for deep learning*, Medium, Augmenting Humanity **16**.

Keizer, Veer I.P., Stefano Coppola, Adriaan B. Houtsmuller, Bart Geverts, Martin E. Van Royen, Thomas Schmidt, and Marcel J.M. Schaaf. 2019. *Repetitive switching between DNA-binding modes enables target finding by the glucocorticoid receptor*, Journal of Cell Science **132**, no. 5.

Maier, Andreas, Christopher Syben, Tobias Lasser, and Christian Riess. 2019. *A gentle introduction to deep learning in medical image processing*, Zeitschrift für Medizinische Physik **29**, no. 2, 86–101.

Osinga, Douwe. 2018. *Deep Learning Cookbook*, 1st ed. (Rachel Roumeliotis and Jeff Bleiel, eds.), O'Reilly Media, Sebastopol, CA.

Sanderson, Grant. 2018. *3 Blue 1 Brown*.

6 Appendix

In order to provide reproducibility and a starting point for further experimentation. Code will be presented as jupyter notebooks that can be found in `https://github.com/DaanVel/Diffusions`.