Machine Learning for Noise Removal in Time-Resolved Data-Sparse Charged Particle Detection

MPhys Physics Year 4 Research Project (B20AX)

Heriot-Watt University

Submission date: 06/05/2022

Student: Ross Anderson (H00295702)

Supervisor: Prof. D. Townsend

Word Count: 6602

Contents

Abstract

Due to the event counting nature of charged particle imaging experiments, Poissonian noise can distort data. Traditional means for removal of this noise typically come with the restriction of vast computational expense and high runtimes. In this project, an alternative approach to statistical denoising was implemented in the form of machine learning. A modified autoencoder convolutional neural network was used to perform denoising on simulated 1D transient signals which originate from time-resolved charged particle detection experiments. A fitting model was then used to retrieve the original time constant(s) used in simulation of this data. The results show that the network can perform near perfect reconstruction of the original noise-free function. Additionally, extracted values for time constants are far more accurate to their original truth value once noisy data is passed through the neural network. This remains true in all cases of low (10%), medium (25%) and high (50%) noise tested. The network does, however, begin to introduce a systematic error when reconstructing transient curves consisting of high time constants. This occurs with values greater than around 1.75 ps for medium noise, and around 1 ps for high noise. Due to this, a limit can be imposed on the highest time constant present in a transient decay that the network can reliably denoise within the constraints of this project. This problem is suspected to have arised due to the relatively short time domain that was used as the temporal axis in simulation of transient decays. Additional insights were also introduced with regard to the ill-posed problem of biexponential decays, with future work in this area forming the basis for further studies.

Acknowledgements

I would to like to thank my primary supervisor, Prof. Dave Townsend, for his support and guidance throughout this project. Specifically, for his patience in explaining many concepts that were previously unknown to me, as well as dedicating time to review my writing many times throughout the academic year.

I would also like to extend my gratitude to my secondary supervisor, Chris Sparling, who helped tremendously with the technical element of this project, on top of also dedicating additional time throughout the year to discuss with me many concepts included in this report.

Declaration of Own Work

Course code and name:	B20AX – Research Project
Type of assessment:	**Individual**
Coursework Title:	Project Report
Student Name:	Ross Anderson
Student ID Number:	H00295702

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.

- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penalties that I will face should I not adhere to the University Regulations.

- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism

Student Signature: *Ross Anderson*

Date: *06/05/2022*

Copy this page and insert it into your coursework file in front of your title page.
For group assessment each group member must sign a separate form and all forms must be included with the group submission.

Your work will not be marked if a signed copy of this form is not included with your submission.

1. Introduction

1.1. Time-resolved charged particle imaging

Charged particle imaging is an area of great importance in the chemical dynamics community, where the intent is to access both energy- and angle-resolved information simultaneously[1]. Electrons or atomic/molecular fragments produced *via* interactions between molecules and laser pulses will have a full recoil distribution which can be projected onto a 2D detection plane using electrostatic lenses. Through analysis of this projected image (an example of which is shown in Fig. 1(a)), one can learn about the specific processes relating to the dynamics of excess energy redistribution in a molecular system following the absorption of a photon, as well as the different ways in which chemical bonds can break. Charged particle imaging studies of molecular dynamics have vast implications in modern science, for example in areas such as biology (i.e. light-harvesting, photo-protection), atmospheric and interstellar photochemistry, rational design photochromic polymers and photostabilizers, sunscreens, molecular switches, and drugs for use in photodynamic therapy[2, 3, 4].

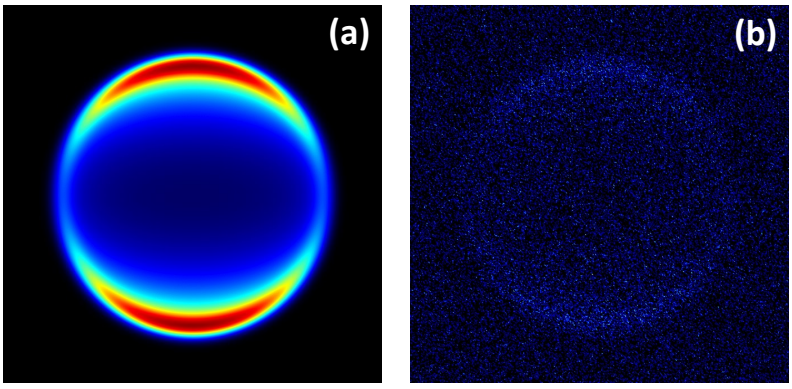


Figure 1. An example of a simulated 2D charged particle projection both noise-free (a) and with an added noise filter (b). This noise filter results in an image with an average of 2.5 counts per pixel, and a signal-to-noise ratio of 10.

A prominent issue in charged particle imaging arises in the form of Poissonian noise. To illustrate this issue, Fig. 1(b) displays a version of Fig. 1(a) with a noise filter introduced. In a standard imaging setup, in which digital cameras containing CCD or CMOS image sensors are widely used, each pixel collects roughly around 10^9 photons per second per pixel[5], such that the variance in number of photons collected

per unit time varies infinitesimally in comparison to the total amount per pixel. If we instead consider very low levels of particle counts, such as in the data-sparse imaging of ions or electrons, we can investigate how this may introduce issues. In charged particle imaging, a velocity map imaging (VMI) system[6] is typically used to focus ions or electrons onto a position sensitive detector. An electrostatic lens is used to accomplish this, having the advantage that ions/electrons with the same velocity are focussed to a single spot on the detector regardless of their original position. As an incoming charged particle strikes the detector, multiple electrons are ejected due to a high voltage being applied across the detector. As shown in Fig. 2, these electrons are accelerated towards a phosphor screen, which in turn excites the electrons in the phosphor. As these excited electrons drop back down to a lower energy level, the excess energy is emitted as photons. These photons are then captured using a CCD imaging sensor, analogous to imaging with a digital camera.

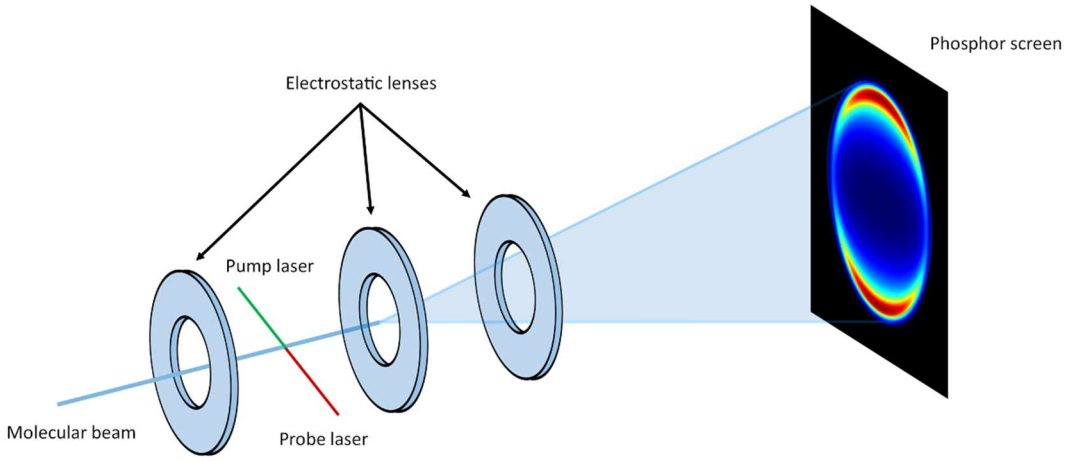


Figure 2. Schematic setup of a velocity map imaging system. Electrostatic lenses are used to focus ions produced *via* interactions with pump-probe laser system onto a phosphor sceen. This excites electrons in the phosphor and causes them to emit photons, giving rise to a 2D projection of the full 3D ion cloud originally produced.

In the regime of low charged particle counts (and therefore low photon counts), the fluctuations in photons detected per unit time can become quite significant, giving rise to Poissonian noise[7]. Another field where this is easily seen is in single photon imaging, which has applications of significant importance in the scientific community, including biomedical studies, neuroscience and cancer diagnosis[8] just to name

a few. With this being an ongoing problem in multiple fields, it seems obvious that developing a means to remove this noise would be advantageous.

Traditional computational solutions for the removal of Poissonian noise from image data usually involve the use of an algorithm that aims to minimise a cost function through the use of a Poissonian likelihood term coupled with a regularization term. The noise-free dataset will be the most probable intensity map given the original image data and regularization restrictions[9]. While traditional methods can indeed work very well and produce excellent data, they come at the cost of high computational expense and experience extreme runtimes[10, 11]. One novel strategy for providing an alternative solution to statistical de-noising can be found in the fast-developing field of artificial intelligence, more specifically, the subset of machine learning.

1.2. Previous work and project direction

Previous work performed in this area undertaken by the Townsend group has involved using machine learning strategies to de-noise individual charged particle images and extract relevant information with success[12]. An example of this previous work is shown in Fig. 3, which demonstrates denoising performed on the data-sparse image shown in (a) to result in (b), a denoised reconstruction of the original image. The corresponding truth image is shown in (c), where "truth" refers to the original noise-free simulation of the image prior to a noise filter being added. Similarities between the reconstruction and the original version of this image show the potential for denoising using machine learning strategies.

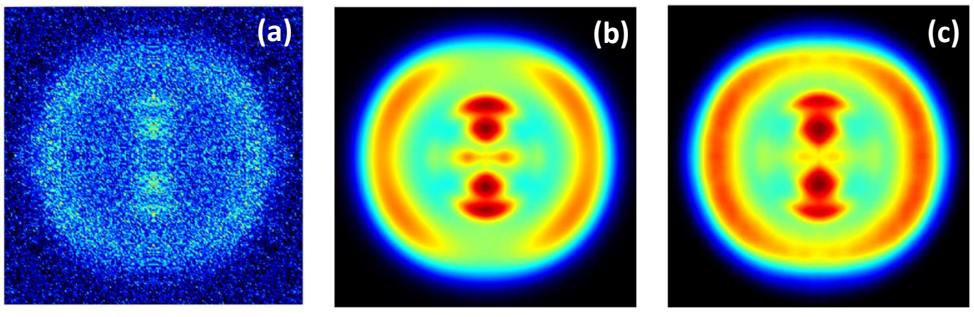
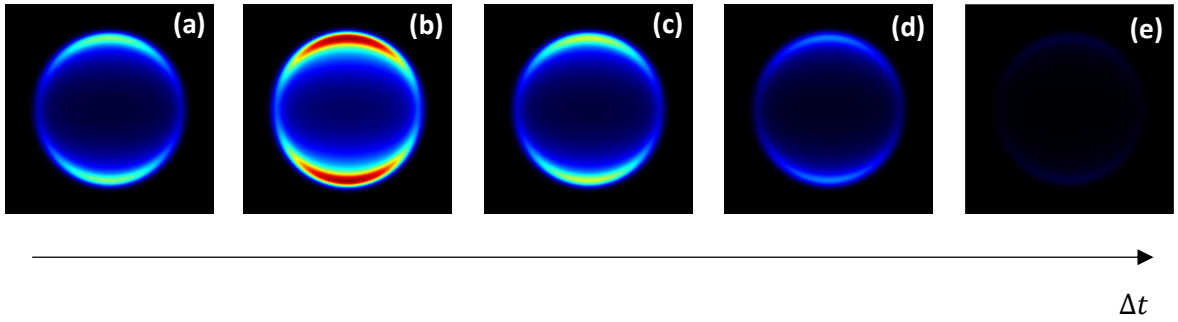


Figure 3. Example of denoising performed on a 2D charged particle projection The data-sparse input (a) is input to the trained machine learning system, resulting in (b), a reconstruction of the original image (c). (taken from Sparling *et al.*[12])

The experiments in which these images are obtained are highly information rich, and therefore this project focuses in on a specific area; the temporal evolution of the atomic/molecular system after excitation. Instead of the purely spatial 2D problem previously considered, this project instead considers a changing intensity over time. Figure 4 shows an example of how these so-called transient decays are obtained by energy-integrating images at different timesteps. The exact mathematical description of the function seen will be covered in Section 2.1., but importantly at this point, this includes an exponential decay. Transient decays have a particular lifetime (or set of lifetimes) associated with them, governed by a single or multiple time constants, which can provide information relating to the dynamics of the atomic/molecular sample being probed.



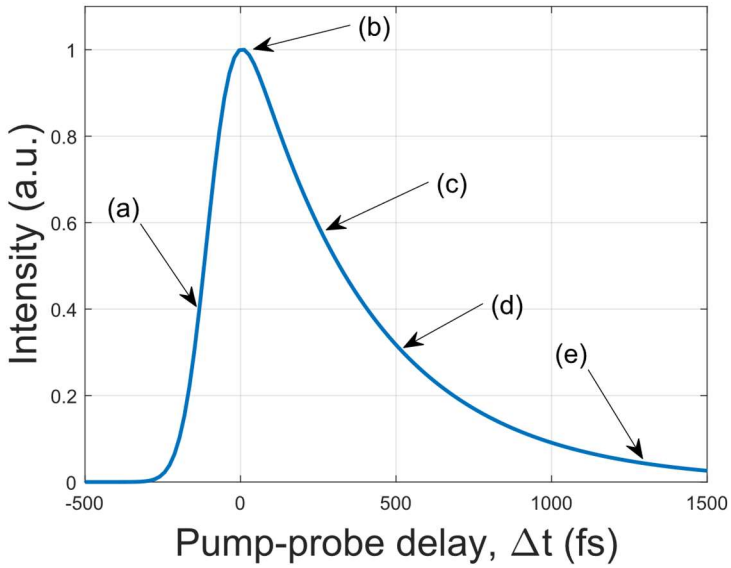


Figure 4. Origin of a transient decay. Charged particle imaging experiments provide (a)-(e), 2D projections of a full 3D ion distribution. The intensity of these images evolves as the pump-probe delay increases. By integrating each image over all energies, the transient decay shown can be produced.

With the knowledge that the transient decay shown in Fig. 4 is obtained by integrating a series of images over all energies (i.e. all pixels), one can start to see that when there is significant Poissonian noise present, the time constant(s) associated with these transient decays will become difficult to extract reliably. Therefore, the overall aim of this project is to build, train and use a neural network to denoise randomly generated transiently evolving 1D signals containing a constituent exponential decay, and subsequently extract the time constant(s) of that decay. This project will contribute as part of a longer term aim to combine 1D denoising in the temporal domain with the spatial 2D image denoising problem previously worked on, so as to work simultaneously to denoise many of these images with changing intensities over time.

1.3. Artificial neural networks

The concept of machine learning is the development of software that will learn and adapt, in a way similar to how humans do, with additional experience advancing knowledge and allowing for greater potential. Machine learning has many current uses in our daily lives, including image classification, recommendation systems, speech recognition, weather forecasting, traffic predictions and parking availability predictions[13] while also being a strong contender for the future, in areas such as autonomous driving[14].

Using traditional computational techniques, a user submits to a computer an input dataset as well as what to do with it, then the computer returns an output with the predetermined algorithm executed on the input. Machine learning differs here, as instead of submitting an algorithm, the user submits both an input dataset and an output dataset, letting the computer return the algorithm that transforms one into the other[15]. There are many different types of machine learning, each specific to the issue at hand. Four typical methods are supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning.

Firstly, supervised learning works by separating and labelling a dataset into training and validation components. The former consists of pairs of datasets, the input and corresponding expected output. These pairs are then used to train a machine learning system to develop a model of the relationship between them. By then applying the algorithm developed in the training process to validation data (defined as a separate stack of data the system has not previously seen), the performance can be evaluated by checking the network output against the expected (i.e. truth) output. This is repeated many times, allowing the system to minimise errors and correct the output accordingly[16]. Supervised learning is mainly used to create a system that is able to extrapolate data to predict outcomes.

By contrast, a system using unsupervised learning takes an input of only an unlabelled dataset, with no knowledge of what corresponds to the training or validation data. The system then attempts to find patterns in the dataset to identify and label training and validation sets itself. A primary issue with unsupervised learning is that it is not always possible for the system to identify the correct pattern in the data, and the dataset may instead be incorrectly separated and labelled. On the other hand, unsupervised learning can also result in the identification of patterns in a dataset that were unknown to exist prior to the system results indicating so[17].

The problem with unsupervised learning can be solved to some extent with semi-supervised learning, a cross between the supervised and unsupervised approaches. Using this method, a dataset is input yet again, but this time a small section is labelled correctly. The system performs supervised learning on this small section of data, and then attempts to use this extra knowledge to extend this to the rest of the dataset. This method can provide the system with a better idea of what patterns to identify but could in theory prevent the system from finding any unknown patterns[18].

Reinforcement learning is often referred to as the most advanced, and complicated, method of machine learning. Unlike supervised or unsupervised learning, a system utilising this technique continuously learns, and is not satisfied by a definite endpoint. Instead of working based off of a specific goal, this type of system instead measures the reward obtained by performing a specific task.

The system will aim to maximise this reward signal, so if it performs a task incorrectly, this will correspond to a low (or even negative) reward, and thus the system will not prioritise the method. By contrast, if the system receives a high reward signal, the system will prioritise the method. For example, with an autonomous driving system, if a manoeuvre is made which successfully avoids a crash, this would reap a high reward signal. Whereas, in a scenario in which the crash is not avoided, clearly this would return a low reward signal[19]. These cases would be simulated infinitely, in order to attempt to cover all possibilities[17, 19, 20].

An artificial neural network (ANN) aims to create artificial intelligence by building a system which has similar architecture to that of a biological organism. Although this analogy makes sense for the origin of the term 'artificial neural network', one may find it easier to understand an ANN in terms of graphing. The ANN takes an input and output dataset, and aims to find the exact parameters which minimise the difference, or loss, between the output of the ANN and the intended output[21].

There are many different types of neural networks which operate in different ways to achieve specific outcomes. Some of the most popular and prevalent architectures are feed forward networks, recurrent neural networks, and convolutional neural networks. Before discussing complicated structures like these, however, the basics of neural networks should first be covered.

The perceptron is the most basic neural network architecture, shown in Fig. 5. It acts as the building block for more complicated systems, consisting of a simple structure containing just a single computational layer, whereas more complicated neural network structures involve multiple layers.

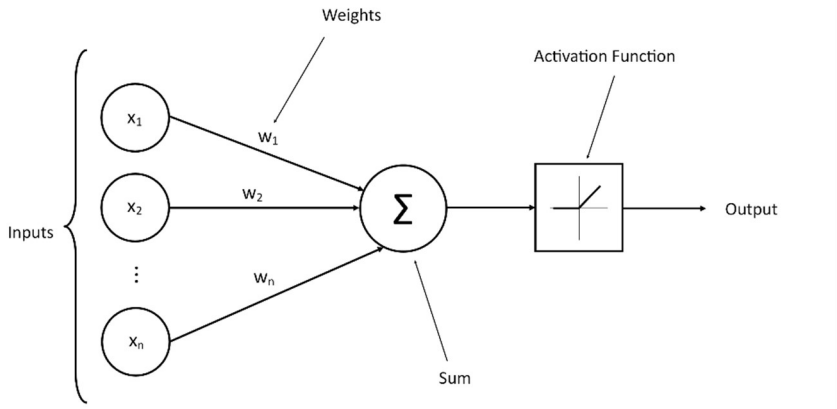


Figure 5. Diagram of a perceptron, a simplified model of a biological neuron. Data flows left to right, with each element of the input vector manipulated by a weighting before being summed and modified again by a specific activation function.

Each of the inputs are designated a specific weighting, multiplied by said weighting, summed, then passed through an activation function. The activation function converts this weighting into a meaningful representation[21], for example, some can set an input to either 0 or 1, depending on whether or not it passes a certain threshold. This could be useful in instances where one may want to use Boolean algebra on logic gates, or just simply generate binary values.

Advancing slightly, multilayer neural networks contain more than one computational layer, but only the final layer (i.e. the output), is visible to the user, the rest are known as hidden layers. These additional layers allow for a greater number of adjustable parameters, which permit the neural network to learn in much more detail. Feedforward neural networks are the simplest example of a multilayer network, they are simply a multilayer perceptron. All data travels in a single direction in this architecture, sequentially, from layer to layer. A feedforward neural network is typically used to learn the relationship between independent and dependent variables, with the former being used as the inputs of the network, and the latter as the outputs[22].

Recurrent neural networks (RNNs) have a similar architecture to a feedforward network, but in an RNN, the layers can include feedback loops, where the result of a layer is fed back into the layer it originated from, creating a 'short-term memory' in the network. The output of the layer in the first iteration can affect the output of the same layer for the next iteration. This design allows for greater ability to predict outcomes based on the previous outcome, with applications in areas where data is sequential or time dependent, such as in speech recognition or predictive text[23].

Convolutional neural networks (CNNs) are a type of feedforward neural network; however, convolution and pooling techniques are applied at each layer in order to transform the dimensionality of the input and identify the primary features. By using a 2D dataset such as an image, convolution can be done by evaluating the dot product between a selection of the image and a randomly chosen filter of the same size, whose values are altered by the network automatically and improved upon as it iterates. Figure 6 displays an example of using a convolutional layer. By performing convolution, filters can be found that allow a certain transformation to be applied to an image, such as identifying particular vertical or horizonal edges of specific features[24]. This is often paired with max-pooling layers, which, shown in Fig. 7, can compress the image while still retaining the important features. These techniques can be used to compose the architecture of an autoencoder neural network. A conventional autoencoder[25] encodes its input by performing dimensionality reduction (a concept that will be discussed in later sections), then learns how to reconstruct its input from said encoded representation.

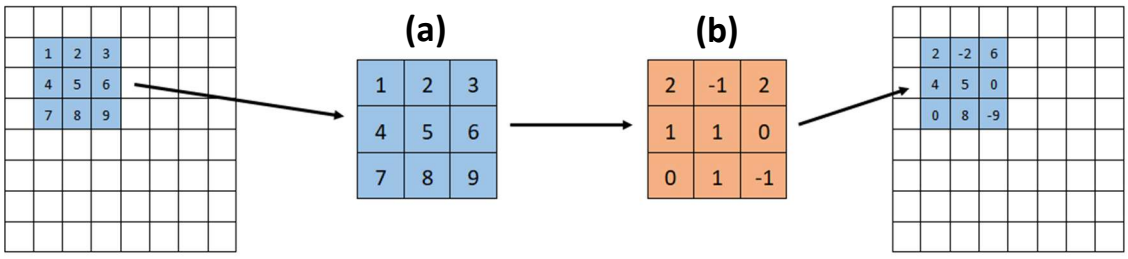


Figure 6. Schematic of a convolutional filter applied to a 2D dataset. (a) indicates a selection of values from this data, while (b) represents a randomly generated convolutional filter. The dot product between (a) and (b) is taken and replaces the location of the original data in the new encoded dataset.

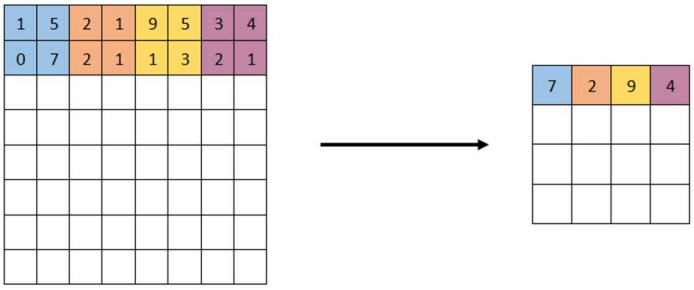


Figure 7. Depiction of a (2x2) max pooling operation on a 2D dataset. The dataset is grouped into smaller 2x2 sections, and the largest value from these sets (differentiated by colour) are kept and included in a compressed version of the original 2D dataset.

In this project a modification will be made to the typical autoencoder by using supervised learning to train. The now modified autoencoder ANN will take a noisy input, perform convolution and pooling techniques to transform the dimensionality of the dataset into an encoded form, where it can be manipulated by weightings. This form is then decoded back to the dimensions of the original dataset. The ANN learns and predicts how to minimise the root mean squared error (RMSE) between the network output and the truth. This is repeated for many different sets of data, and over time this difference will converge. With training complete, previously unseen noisy data can be given to the ANN and a noise-free version containing the true structure will be produced.

2. Materials and Methods

As discussed in the introduction, an ANN needs to be trained first in order to be used. A specific training dataset is therefore required to perform this, which should aim to include as many situations as possible that the network might see. To generate entire training sets, a script capable of mass-producing completely randomised transients must be created. To accomplish this, a more in-depth investigation into the origin of these signals is required.

2.1. Introduction to ultrafast spectroscopy

Ultrafast spectroscopy allows one to measure the temporal evolution of a system after excitation. This is done by introducing two laser pulses: the pump and the probe. The pump prepares an atom/molecule in an excited state, while the probe subsequently interrogates this excited state population after some adjustable time, known as the pump-probe delay, Δt. An example of the shape and origin of the evolving signal is shown in Fig. 8.

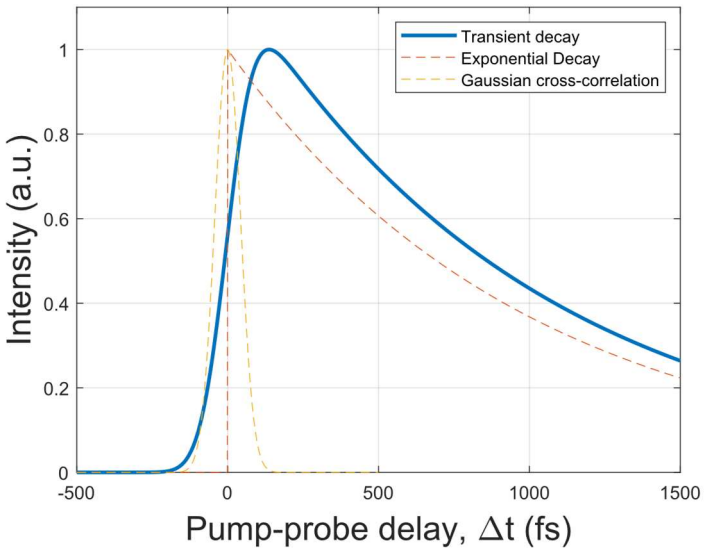


Figure 8. Example simulated as part of this project using MATLAB of a transient decay. This results from a convolution between two functions shown as dotted lines: A Gaussian cross-correlation (with a FWHM of 100 fs) between the pump and probe laser pulses and an exponentially decaying function starting at Δt=0.

This shape is described by a mathematical convolution of the Gaussian cross-correlation between the pump and the probe and an exponential decay starting at $\Delta t = 0$, as seen by the constituent functions shown as dotted lines in Fig. 8. The Gaussian cross-correlation is described by equation (1), where t represents the time domain over which the pulse exists and σ is the standard deviation. Equation (2) allows a substitution to be made for σ in order to represent the Gaussian pulse in terms of the full width at half maximum (FWHM) rather than the standard deviation. This is done to appeal to the common standard in the chemical dynamics community. Additionally, throughout each transient, the FWHM was kept constant at 100 fs, as this was seen as a typical value used experimentally. The exponential decay is shown in equation (3), which portrays an exponentially decaying function where Δt is the pump-probe delay, τ represents the time constant, and $H(\Delta t)$ represents the Heaviside step function, shown in equation (4). The convolution between equations (1) and (3) is shown in equation (5).

$$f(t) = \exp\left(-\frac{t^2}{2\sigma^2}\right) \tag{1}$$

$$\sigma = \frac{FHWM}{2\sqrt{2\ln(2)}} \tag{2}$$

$$g(\Delta t) = \exp\left(-\frac{\Delta t}{\tau}\right) H(\Delta t) \tag{3}$$

$$H(t) = \begin{cases} 1, & t > 0 \\ 0, & t \leq 0 \end{cases} \tag{4}$$

$$(f * g)(\Delta t) = I_{single} = \int_{-\infty}^{\infty} f(t) g(\Delta t - t)\, dt \tag{5}$$

For an analytical alternative, we can explicitly compute the result of this convolution[26], shown in equation (6). As learned throughout the project, there are significant advantages to working with an analytical

solution, rather than numerical, as will be discussed in Section 2.4. As before, one can also use equation (2) to rewrite this equation in terms of the FWHM rather than standard deviation if one chooses to.

$$I_{single} = \frac{1}{2}\exp\left(-\frac{1}{\tau}\left(t - \frac{\frac{1}{\tau}\sigma^2}{2}\right)\right)\left(1 + \operatorname{erf}\left(\frac{t - \frac{1}{\tau}\sigma^2}{\sqrt{2}\,\sigma}\right)\right) \tag{6}$$

Focussing on the exponential decay aspect of the convolution (given by equation (3)), we can introduce additional functions to describe more complex, multi-step transient behaviour. Coupling individual decays together can be done either in parallel or sequentially. Parallel dynamics corresponds to decay processes acting independent of one another, whereas sequential dynamics corresponds to a system where the second decay term grows as the first falls, then declines according to its time constant. Figure 9 portrays the different types of decay alongside the transient that results from their convolution with the Gaussian cross-correlation function.

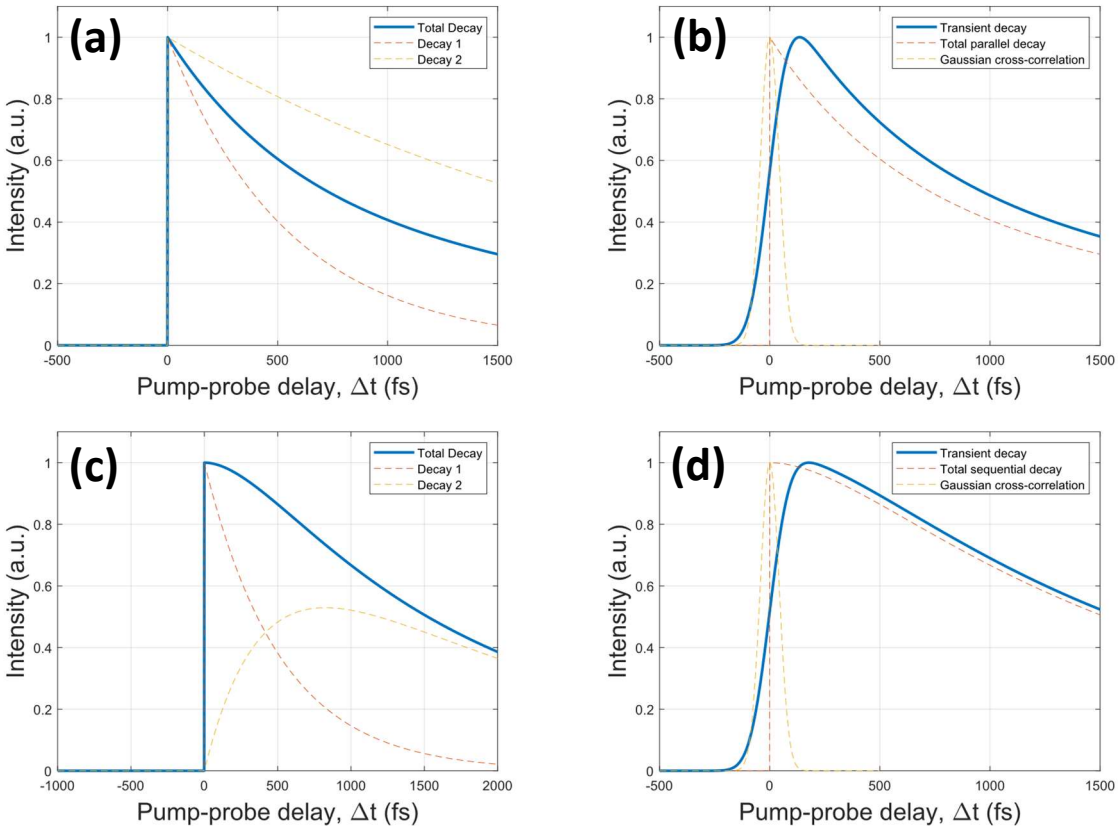


Figure 9. Examples simulated as part of this project using MATLAB illustrating the differences between parallel and sequential decays. (a) displays parallel dynamics, where two decays begin at the same point and decline according to different time constants. (b) demonstrates sequential dynamics, where the second decay rises as the first falls, then decays according to its own time constant. (c) & (d) show the total decays convolved with the Gaussian cross-correlation of 100 fs, for parallel (a) and sequential (b) dynamics respectively.

2.2. Simulation of experimental data

As shown in at the start of Section 2.1, the convolution between a Gaussian cross-correlation and exponential decaying function results in a transient decay. The initial goal was to create a script capable of randomly simulating sufficient transient curves to satisfy the training data requirements. This was done by randomising the time constant, and generating 1,000 curves different per training set. Full details of the code produced can be found in Appendix 3: Training Data. As previously mentioned, the FWHM of the Gaussian cross-correlation was kept constant at 100 fs throughout the process. The time constant was chosen by using the MATLAB "randi" function to randomly select an integer between 50 fs and 2,000 fs each time a new transient curve was to be simulated. These limitations on the time constants permitted were viewed as typical values seen experimentally. With the random aspect of the simulation

working well, the next step was to introduce some noise. To accomplish this, the training data had to first be made as similar to experimental data as possible. Data points were extracted at every 50 fs, a typical timestep used in time-resolved ultrafast spectroscopy experiments. By extracting values at every 50 fs from a dataset ranging from -500 fs to 1,500 fs as shown in Fig. 10(a), this resulted in an array of 41 points. In practise, the final data point was removed so that the temporal range was instead between -500 fs and 1,450 fs. This was changed in order to have an even number of points in each transient, for later benefits in the dimensionality alteration done by the ANN. The intensity values corresponding to these points were randomly assigned a set percentage of noise to be added or subtracted. In the example shown in Fig. 10(b), a maximum possible value of 25% noise was incorporated into the model.

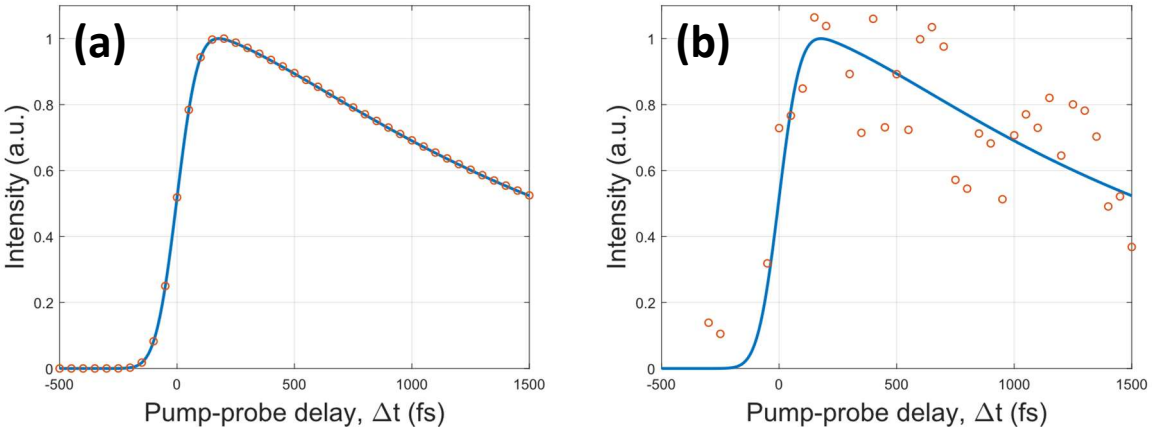


Figure 10. Example simulated as part of this project using MATLAB showing the extraction of data points and subsequent addition/subtraction of noise. (a) portrays the continuous transient function familiarised in Section 2.1. (shown in blue), with data points taken at every 50 fs (shown in red). (b) indicates once again the discrete transient function (shown in blue), along with extracted data points with up to 25% added noise (shown in red)

2.3. Designing and implementing the architecture of the ANN

Upon completion of the training set, the design of the ANN was the next objective to complete. The TensorFlow and Keras Python libraries were used to provide an interface for the ANN, and all execution of the code produced was done on Google Colab, a Python integrated development environment which provides free processing use *via* cloud services. Google Colab was used in order to avoid having to use personal equipment for computationally expensive procedures. Figure 11 displays visually the full neural network architecture used in this project, and this section will discuss each step involved, however, for full details of the neural network code produced for this project, see Appendix 4: Artificial Neural Network.

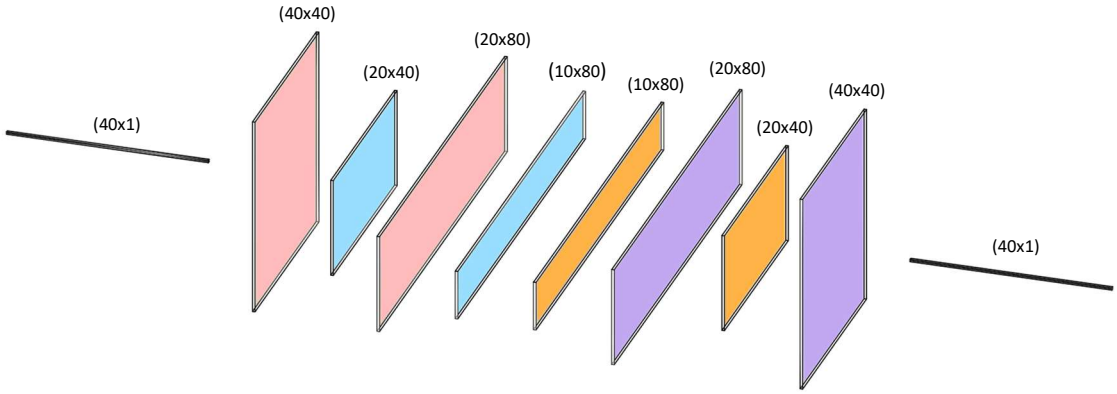


Figure 11. Graphic produced using an online tool which shows autoencoder ANN architecture utilised in this project for denoising. The data flows from left to right, with either a convolution (red), max-pooling (blue), deconvolution (orange), or concatenation (purple) operation occurring at each layer. These operations alter the dimensionality of the data into a complex form where it can be manipulated by a filter matrix consisting of many different weightings.

As seen in Fig. 11, as the input data flows from left to right, the dimensionality of the data is manipulated. This is done by either a convolution, max-pooling, deconvolution, or concatenation operation, as well as the use of the ReLU activation function[27] at each stage. The first step is a convolution between the input data consisting of a (40,1) vector - which contains data points from -500 fs to 1,450 fs on the transient curves seen in previous sections – and a (40,40) filter. This results in the dimensionality of the data being converted to (40,40), resulting in 1,640 trainable parameters. Next, a 1D max-pooling operation with a pool size of 2 is computed on the new (40,40) matrix, converting the data to a (20,40) dimensionality. After that, another convolution between this (20,40) matrix and a (40,80) filter is performed, resulting in the data being transformed to a (20,80) form and forming 128,080 trainable parameters. Then, another 1D max-pooling with a pool size of 2 is computed, resulting in a (10,80) form, this is the centre part of the network shown in Fig. 11, called the 'bottleneck' in machine learning terms. Finally, the exact opposite steps are done to return the data back to the original (40,1) form, using concatenations and deconvolutions in place of max-pooling and convolutions. Overall, this results in a total of 515,441 trainable parameters in the network.

2.4. Training the ANN with simulated data

Now that the network architecture has been presented, the exact methods of training can be discussed. With the large dataset consisting of 1,000 different curves with time constants between 50 fs and 2,000 fs established with uniform probability, this dataset is first split into two subsets, 90% is kept for training and the remaining 10% for validation. The network takes the input noisy transients through the layers, and begins to make a prediction as to the original noise-free versions. The first few attempts will more than likely produce a poor result, as the network has yet to learn. This is where the concept of supervised learning comes to light. The network's output is compared against the corresponding truth data, and the network quantifies this difference using the RMSE, shown in equation (7), where $\hat{y}_t$ is the predicted value of y_t, with these variables observed T times.

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}} \tag{7}$$

Each iteration, the network attempts to reduce the RMSE between its prediction and the truth. Once this has been done for each curve within the training set, this is averaged as the overall loss for this total iteration over all input data. In machine learning terms, this complete iteration over an entire dataset is defined as an 'epoch'. Additionally, at the end of each epoch, the network code written for this project will also show the user the RMSE for the performance on validation data at this point. The network does not actively attempt to minimise this, it is instead just a metric for the user to follow. This process is repeated in this network until 3,000 epochs, at which the RMSE has completely converged to a very small value, varying for different noise levels, but generally in the range of 10^{-4}-10^{-7} in this project. It is worth mentioning now that each noise level used in this project consists of its own training set, and its own trained network. A network trained on low noise levels would perform poorly when introduced to high levels of noise, so a new network is trained each time.

2.5. Extraction of time constant in single-decay data

Now that the procedures of the autoencoder ANN used in this project have been identified and explained, the next steps can be discussed. As stated in the introduction, the purpose behind aspiring to successfully denoise 1D transient curves is to be able to identify the time constant that corresponds to the lifetime of the atomic/molecular sample used in ultrafast spectroscopy. In order to do this, fitting to the analytical model of these curves can be employed. For this, MATLAB was used due to its relatively straight forward Curve Fitting Toolbox. A full version of this fitting code produced is provided in Appendix 5: Curve Fitting to 1D Transients. In the case of single-decay data, with the FWHM of the cross correlation known (as discussed in Section 2.1, this was kept constant at 100 fs), the only unknown in this equation is the time constant. Fitting the truth to this model should therefore always output the original time constant that was used to simulate it in the first place. Fitting to the noisy data will produce an attempt at obtaining the original time constant, and can be compared against the known true time constant. The success of the neural network denoising can then be categorised by whether this guess at the original time constant is more or less accurate after the noisy data has been processed through the network.

2.6. Extraction of time constants in multi-decay data

When moving on towards extracting the time constants of transients consisting of multiple decays, this problem becomes increasingly more difficult. Although this project set a limitation on the number of decays to not exceed two, the introduction of additional time constant comes with a variety of issues to unravel. An ongoing issue regarding fitting to biexponential decays is that many possibilities of two curves combined together with their respective amplitudes can produce very similar looking transients. The problem is said to be ill-posed. Additionally, there is no way to establish which is the first decay, and which is the second, making quantifing the difference between the extracted and true values of τ_1 and τ_2 much more difficult. For this reason, in the training data code produced for this project (see Appendix 3: Training Data for more details), a restriction has been placed on τ_2 that it must always be larger than τ_1, so that it can always be established which is τ_1 and which is τ_2. When working with biexponential decays - as detailed at the end of Section 2.1 - these can act either in parallel or sequentially. To describe either

involves different combinations of the analytic transient decay expression shown earlier in equation (6), as detailed below in equation (8) for parallel dynamics, and equation (9) for sequential dynamics, where A_1 and A_2 denote the amplitude used for the first and second decay respectively.

$$I_{single}(\tau) = \frac{1}{2}\exp\left(-\frac{1}{\tau}\left(t - \frac{\frac{1}{\tau}\sigma^2}{2}\right)\right)\left(1 + \operatorname{erf}\left(\frac{t - \frac{1}{\tau}\sigma^2}{\sqrt{2}\,\sigma}\right)\right) \tag{6}$$

$$I_{parallel}(\tau_1, \tau_2) = A_1 I_{single}(\tau_1) + A_2 I_{single}(\tau_2) \tag{8}$$

$$I_{sequential}(\tau_1, \tau_2) = A_1 I_{single}(\tau_2) + A_2 I_{single}\left(\frac{\tau_1 \tau_2}{(\tau_1 + \tau_2)}\right) \tag{9}$$

When extracting the time constant for biexponenial decay transients, a similar approach was taken as for the single-decay data. Importantly, it was found when working with sequential decays that if first a parallel model was assumed, one of the amplitudes would return a negative value. This was utilised in the time constant extraction code produced (see Appendix 5: Curve Fitting to 1D transients for full details) to act as a method of classifying the different decay behaviour. Once classified, the appropriate analytical expression was used in order to fit a model to simulated data both before and after being denoised by the network.

3. Results

The results of denoising 1D transient signals using the modified autoencoder ANN detailed in Section 2.2 has been split into two parts to separate single and biexponential decay data. Additional care has be taken in the first instance to allow the reader to understand the denoising performed and the display structure of different figures that will be shown.

3.1. Single decay

3.1.1. Denoising of 1D transients

As detailed previously in Section 2.1-2.2, the training data script written as part of this project can mass produce 1D transients both noise-free and with added noise. For each level of noise tested in the following results, a network was individually trained using a dataset produced specific for that noise level. Fig. 12 demonstrates the production of these pairs of noisy and noise-free transients, and shows the results of ANN denoising, overlayed on top of the original noise-free version.

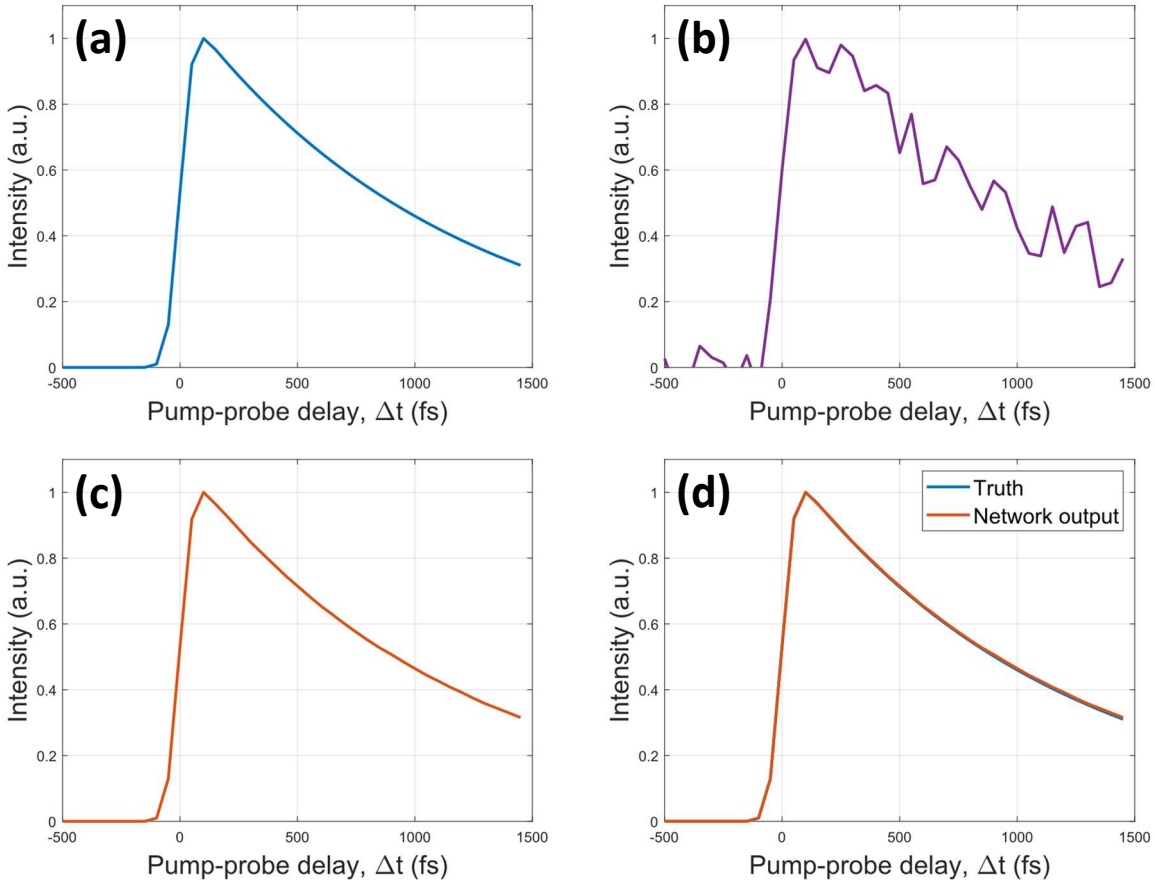


Figure 12. Multiple graphs generated using MATLAB showing a detailed walkthrough of the simulation and denoising process. The truth curve (a) was firstly generated using the training data generation script (Appendix 3: Training Data) detailed in Section 2.1, then random noise between 0-10% was added/subtracted to each data point, resulting in (b). This noisy dataset was passed through the modified autocoder ANN shown in Section 2.3, leading to the network output shown in (c). Overlapping the network output (c) over the original truth curve results in (d).

With the denoising power shown for low levels of noise, Fig. 13 displays both the input noisy transient along with overlapping the network output for different input noise levels (using their respective trained networks) with the corresponding truth, as done in Fig. 12(d).

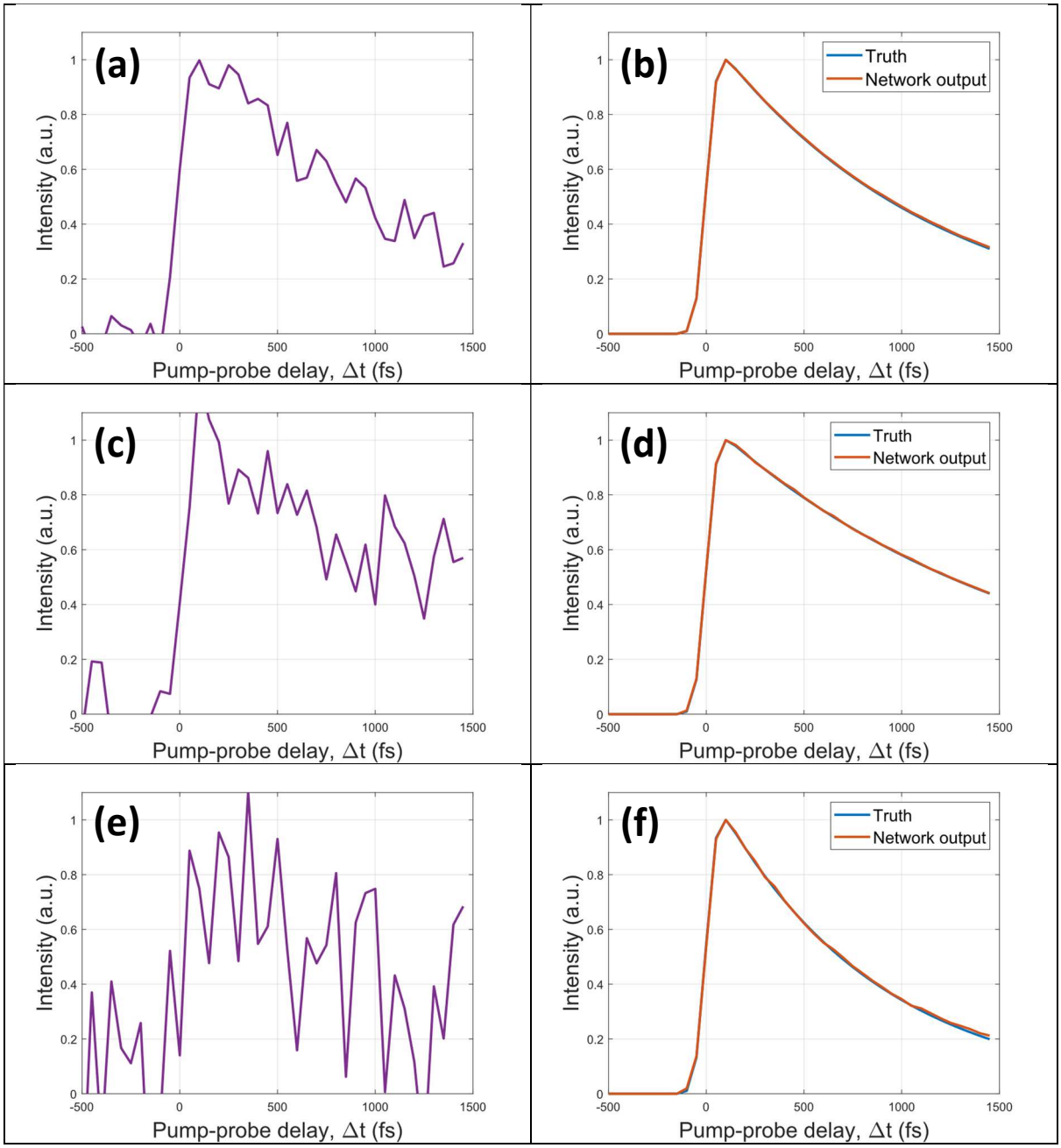


Figure 13. Examples produced as part of this project showing results of network denoising. (a), (c) & (e) correspond to the simulated noisy transient input to the respective neural network for each noise level. (a) corresponds to low (10%) noise, (c) to medium (25%) noise and (e) to high (50%) noise. (b), (d) & (f) show the network output plotted alongside the original noise-free truth curve for low, medium and high noise respectively.

As can be seen from Fig.13, the network can successfully reconstruct the original transient shape prior to a noise filter being added. This remains true for all noise levels tested.

3.1.2. Extraction of time constant

As mentioned previously, the true test of this project was whether the network output could be used to extract the fundamental time constant that governs a transient decay more accurately than by simply using the noisy version. The results of this endeavour are shown in Fig. 14, where the true time constants in the low (10%) noise, medium (25%) noise and high (50%) noise cases shown are 1,911 fs, 1,692 fs and 831 fs respectively. It can be seen that processing the noisy data through the network allows for a much more accurate extraction of time constant from the transient decay, and a dramatic reduction in the associated error.

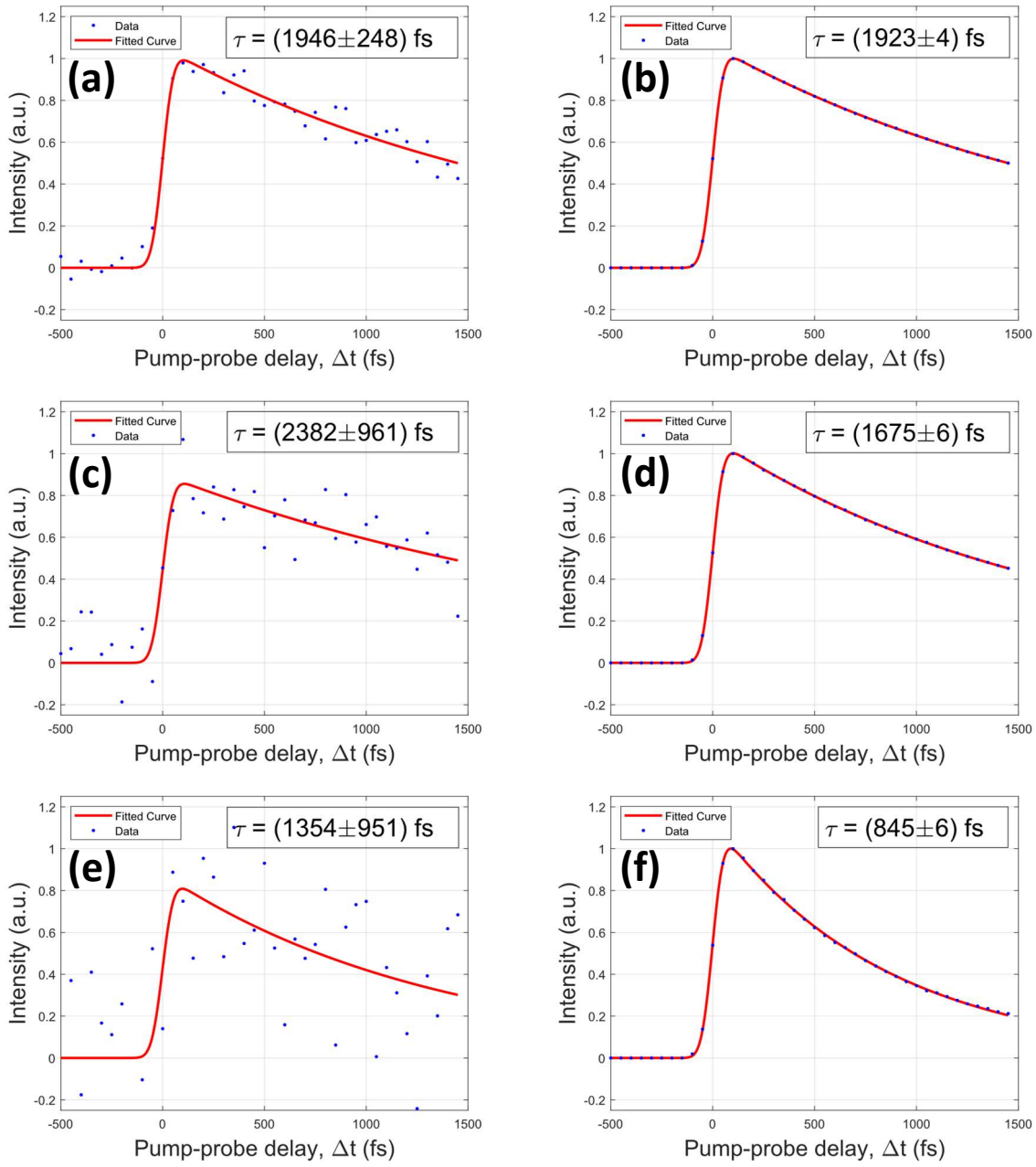


Figure 14. Examples produced as part of this project showing time constant extraction on single-decay transient data. (a), (c) & (e) represent extraction done on low (10%), medium (25%) and high (50%) noise respectively, and (b), (d), & (f) show extraction done after noisy data is processed by the neural network. The true time constant for (a) & (b) was equal to 1,911 fs, for (c) & (d) it was 1,692 fs and for (e) & (f) it was 831 fs.

To express the full accuracy and power of the ANN for different noise levels, large datasets were produced using the training data script written for this project and time constant extraction was computed for each individual curve. Figures 15, 16 & 17 show the results of the extraction of time constant on a large dataset, for low (10%), medium (25%) and high (50%) noise respectively. In each case, the axes depict the extracted time constant plotted against the true time constant. By setting these axes, a straight $y = x$ line (shown in red) can be established to correspond to the truth values. Next, the time constants extracted using the noisy data were plotted (blue points), these represent the time constant found by curve fitting to noisy data (as in the examples shown in Fig. 14(a), (c) & (e)). Each noisy transient corresponding to these time constants was then passed through the network (trained for the respective noise level), and time constant extraction was performed again (orange points).

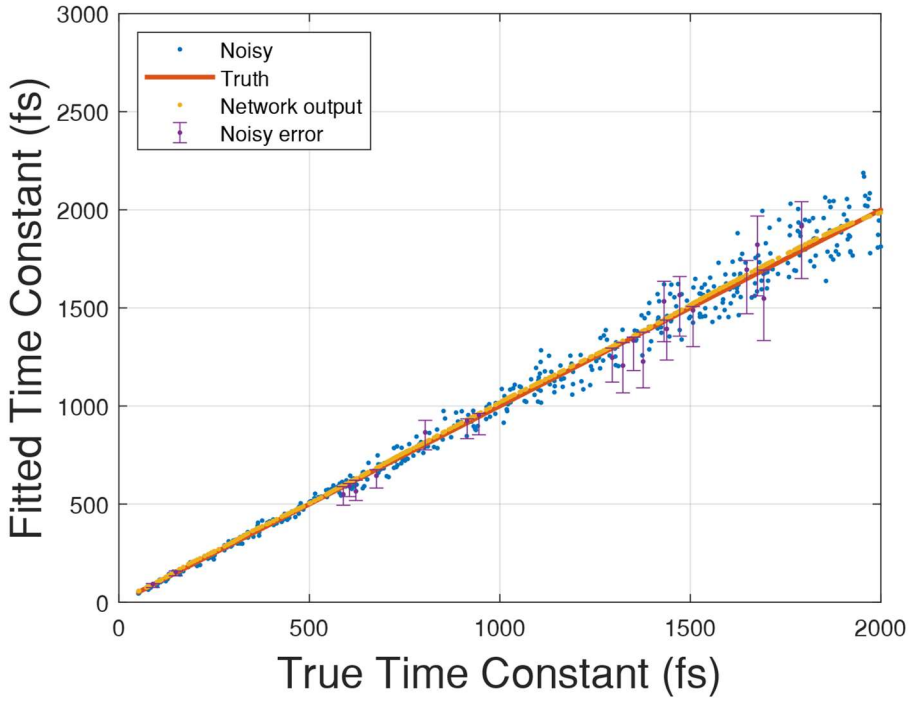


Figure 15. Extraction of time constant performed on large dataset of low (10% noise) transients. Graph depicts a plot of Fitted Time Constant against True Time Constant, permitting the establishment of a y=x straight line corresponding to fitting to truth data (shown in red). Time constant extraction is then performed on both raw noisy data (results shown as blue points) and noisy data processed through the ANN (shown in orange).
Error bars are added (by using MATLAB "confint" function which returns 95% confidence bounds) to show the ANN's ability to reduce the error associated with noisy data (error bars are not shown for network processed data due to being reduced beyond visibility), this will continue for Fig. 16 & 17. Only a small selection of error bars were added so to not clutter the graph.

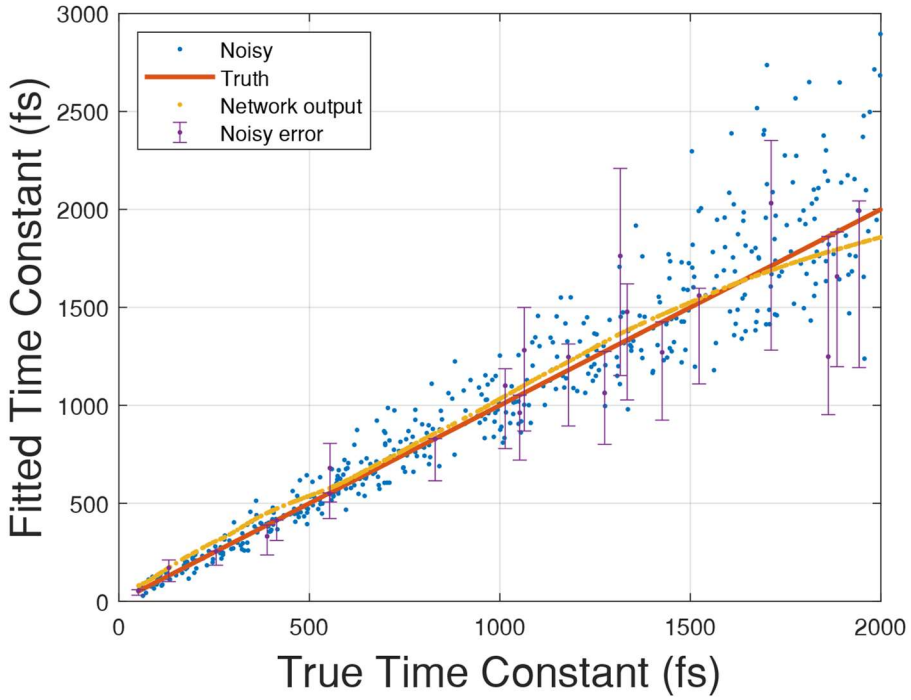


Figure 16. Extraction of time constant performed on a large dataset of medium (25%) noise transients. Identical structure to Fig. 15, see caption for full details. Network output appears to begin to trend away from truth line at time constants exceeding around 1.75 ps, possible causes for this are detailed in the text.

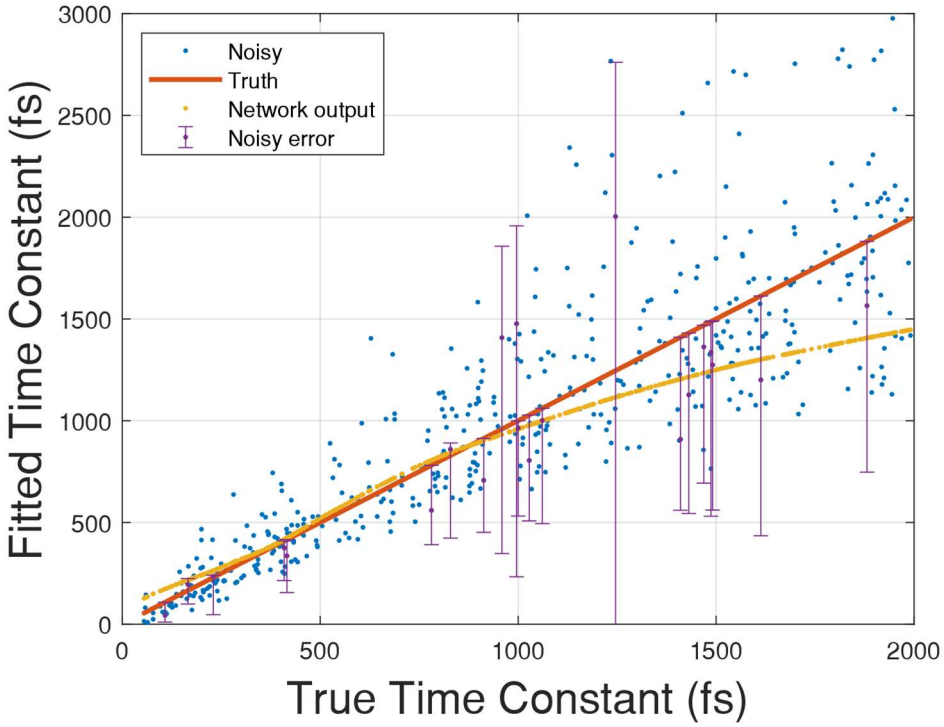


Figure 17. Extraction of time constant performed on a large dataset of high (50%) noise transients. Identical structure to Fig. 15, see caption for full details. Network output appears to trend away from truth line at time constants exceeding around 1 ps, possible causes are detailed within the text.

Figure 15 shows the network performance to be near-perfect at the low (10%) noise level. As the noise levels increases, however - as seen in Fig. 16 & 17 - a limit on the maximum time constant associated with a transient decay at which the network can successfully denoise and subsequently extract is imposed. This limit appears to be around 1.75 ps for the medium (25%) noise level shown in Fig. 16, and around 1 ps for the high (50%) noise shown in Fig. 17. The exact reasoning for this limit at high time constants could be explained by the time domain limits used in the simulated transients. As detailed in Section 2.2, the pump-probe delay is set between -500 fs and 1,450 fs in 50 fs increments to result in a (40x1) dataset. If instead, the maximum was increased to perhaps 3,450 fs (which would result in a (80x1) dataset), at higher time constants datasets would contain additional information to allow the network to better differentiate between curves at similar high time constants.

3.2. Biexponential decay

An additional issue briefly combated was the considerably more complex issue of a biexponential decay. In order to keep this intricacy at a manageable level, this project kept to only introducing one more additional time constant to arrive at a transient decay goverened by two time constants. As previously established in Section 2.2, the key problem in curve fitting to biexponential data is the issue of a transient decays being able to be constructed from many possiblities of time constants and respective amplitudes and still appear identical, an example of this is shown in Fig. 18.

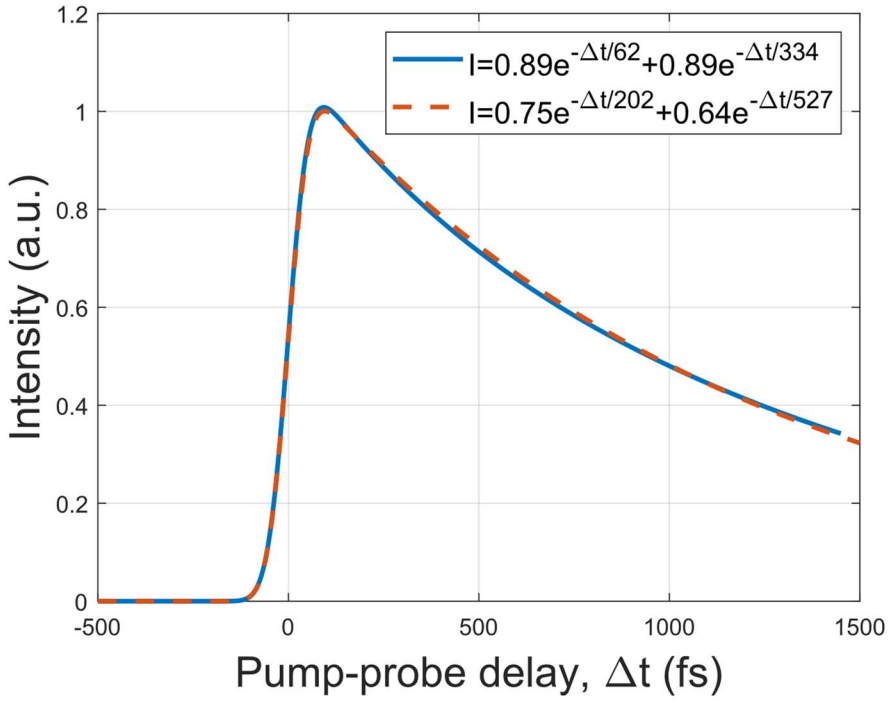


Figure 18. Example of the biexponential problem. The equations for these two transient decays are completely different, although produce very similar looking functions. The blue transient includes a decay with an amplitude and time constant of 0.89 and 62 fs respectively, and a second decay with an amplitude of 0.89 and time constant of 334 fs. The red transient contains a decay with an amplitude of 0.75 and time constant of 202 fs, and a second decay with an amplitude of 0.64 and time constant of 527 fs.

Figure 19 shows examples of time constant extraction using the fitting code produced to tackle biexponential decays. It can be seen that the network can successfully restore the original function shape. It is, however, important to note here that due to being able to construct an extremely similarly shaped function with a variety of time constants and decay amplitudes, the time constant values extracted by the

fitting code may differ from the original used in simulation of these transients. The fitting code has likely succeeded, but the global minimum of best fit is very flat, and consists of many possiblities. The error associated with the time constants extracted is, however, reduced in all cases of low (10%), medium (25%) and high (50%) noise levels examined.

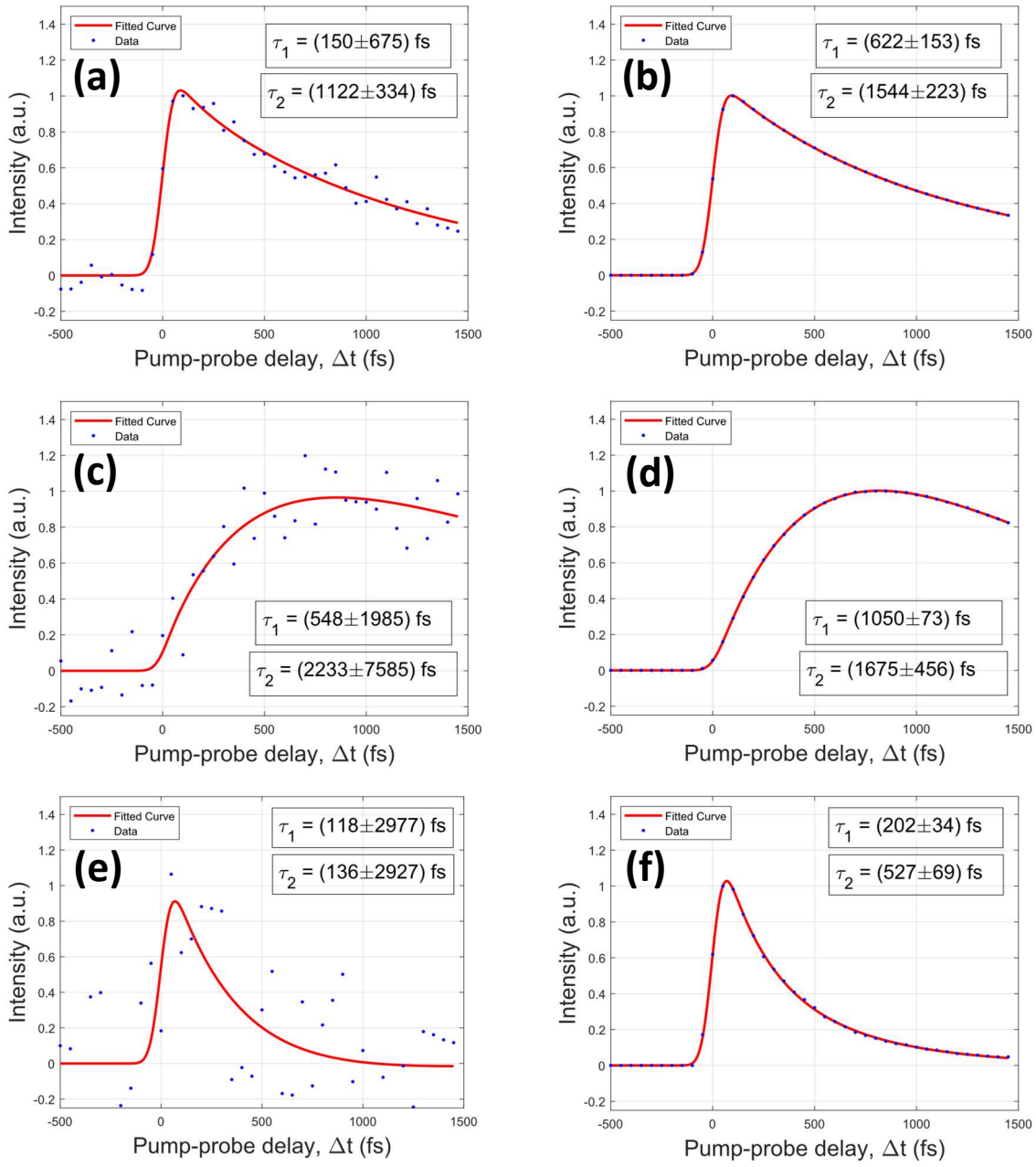


Figure 19. Examples produced as part of this project showing time constant extraction on biexponential decay transient data. (a), (c) & (e) represent extraction done on low (10%), medium (25%) and high (50%) noise respectively, and (b), (d), & (f) show extraction done after noisy data is processed by the neural network.

The true time constants for (a) & (b) were equal to 710 fs and 1,923 fs, for (c) and (d) they were equal to 1,036 fs and 1,921 fs, and for (e) & (f) were equal to 986 fs and 1,507 fs.

4. Discussion

The purpose of this project was to design, train and use a neural network to denoise randomly generated transiently evolving 1D signals, and subsequently extract the time constant(s) which relate to the dynamics of the sample. The long-term goals of this project were to act as part of a larger issue within the field, contributing the time-resolved aspect of a full 3D problem, with the 2D spatial problem previously worked on in the Townsend group. The reasoning behind using machine learning was to attempt to improve upon the high computational expense and long runtimes experienced traditionally through statistical denoising. Many communities would benefit from a faster method of denoising, specifically those who deal with data-sparse experimental results, such as in chemical dynamics or single-photon imaging. The specific intent here was to arrive at a machine learning system which can successfully denoise a collection of comparably similar datasets quicker than traditional means and consequently allow for extraction of the time constant(s) which govern the transient decay using a curve fitting package.

Overall, the results show that for all noise levels considered in this project, the network can successfully restore the original noise-free simulated transient. The noise levels chosen to analyse were low (10%), medium (25%) and high (50%). Low noise was intended as a somewhat simple test, if the network was failing at this level, something would have to be fundamentally changed in the architecture. The medium noise level operated as the main test of this project, meant to capture the real-world possibilities of noise incident in data-sparse experimental results. The high noise level acted as an extreme test for the network, and somewhat surprisingly does continue to denoise successfully in most cases. At higher time constants, the medium and high noise levels can prove slightly too much for the network to handle, and the accuracy of the extracted time constant is decreased. In the medium noise training set, this is seen at roughly around 1.75 ps, towards the tail end of the time constants considered in this project. This effect is seen more easily on the high noise training set where after around 1 ps the performance decreases and a systematic error is introduced. This allows a limit to be established on the ability of the network to denoise and to the fitting code produced to extract the time constant(s), given the constraints used. It is, however, these contraints (particularly the time domain range chosen) that is suspected to be the cause of the performance issues at high time constants. Given that transients comprised of an exponential decay with a high lifetime

will appear very similar when the time domain is far shorter than that lifetime, it seems likely that the network had trouble differentiating these without the additional information greater temporal axis limits would have provided. If this work were to be repeated, or more time was available, it would be recommended to try a time domain that extended the (40x1) array between -500 fs and 1450 fs used to an (80x1) array instead between -500 fs and 3,450 fs, if the same time constant constraints were used.

One motivation for this project was the current state of traditional data processing including high computational expense and slow runtimes. By using machine learning, any computational expense included in denoising is instead all concentrated to the training process. After completion, the network can provide instant on-the-fly denoising for the respective noise level trained. Overall, if one forgoes the training process, using a neural network for denoising is absolutely a faster method of denoising as compared to traditional means. Choosing to instead include the training process, the benefits of using a machine learning system increase each time the network is used. Traditionally the problem has to be solved each time a new dataset is to be denoised, but with a neural network, this is not the case. The time savings of using a neural network to denoise compound as one denoises more and more. The use of large datasets, or an extremely common usage would make training a network to denoise worthwhile.

In regards to the biexponential decay data, this project has investigated and introduced a few problems that still require ongoing investigation to fully understand and solve. The key issue here is that the introduction of an additional time constant as well as relative amplitudes for each individual decay can result in many different configurations of very similar looking functions. When using curve fitting tools, this makes the chosen starting points for the fitting code extremely relevant. Ideally, these would be randomised, but doing so can bring the fitting code too far from the best fit and stop the fit converging completely. This was the final stage of the project, and thus if additional time were available, this problem would have been approached in a more in-depth manner, and would perhaps provide a better understanding of the issue.

5. Conclusions

The approaches used in this project demonstrate the results of using a modified autoencoder artificial neural network to denoise 1D transient decays originating from time-resolved charged particle detection experiments. A fitting code was then produced in order to uncover the original time constant(s) associated with the transient decay, which can provide information relating to the dynamics of the atomic/molecular sample used. The modified autoencoder ANN architecture used transforms the dimensionality of the input data by implementing convolutional and max-pooling techniques, then reinstates the original proportions after they have been modified by trainable parameters in the machine learning system. The results of this project show that the network design used is capable of performing noise removal with high accuracy at the 10%, 25% and 50% noise levels tested. At the 25% and 50% noise levels, however, the network introduces a systematic error when denoising transients of high time constants exceeding 1.75 ps for 25% noise, and 1 ps for 50% noise. The cause of this was suspected to be due to a high similarity between noisy transients at high time constants within the relatively short temporal domain axis range chosen. Finally, additional insights into curve fitting transients consisting of biexponential decays have been established, with results suggesting additional research into this topic should be undertaken in future.

6. Further Work

In regards to this project, an area of possible improvement is the simulation of transients. As discussed, the temporal axis limits chosen (-500 fs to 1,450 fs) are relatively short in comparison to the decay time constants permitted (50 fs to 2,000 fs). Had these axis limits been increased, the performance issues seen when denoising transient decays consisting of higher time constants may be rectified.

Additional work to be done in this field overall includes the longer-term goal of this project; the combination of 1D temporal denoising with previously worked on 2D spatial denoising. Making these work simulateously could be the topic of a future research project within the Townsend group.

Additionally, as spectulated upon in Section 3.2., the biexponential problem appears to be an area which could benefit from additional research.

References

1. Chandler DW, Houston PL, Parker DH. Perspective: Advanced particle imaging. *The Journal of Chemical Physics* **147**, 013601 (2017).

2. Ashfold MNR*, et al.* Imaging the dynamics of gas phase reactions. *Phys Chem Chem Phys* **8**, 26-53 (2006).

3. Gowen AA, O'Donnell CP, Cullen PJ, Bell SEJ. Recent applications of Chemical Imaging to pharmaceutical process monitoring and quality control. *European Journal of Pharmaceutics and Biopharmaceutics* **69**, 10-22 (2008).

4. Svanberg S. Some Medical and Biological Applications of Ultrafast Lasers.). Springer New York (2004).

5. Llull P*, et al.* Coded aperture compressive temporal imaging. *Opt Express* **21**, 10526-10545 (2013).

6. Eppink ATJB, Parker DH. Velocity map imaging of ions and electrons using electrostatic lenses: Application in photoelectron and photofragment ion imaging of molecular oxygen. *Review of Scientific Instruments* **68**, 3477-3484 (1997).

7. Blanter YM, Büttiker M. Shot noise in mesoscopic conductors. *Physics Reports* **336**, 1-166 (2000).

8. Yu Chen YC*, et al.* Recent advances in two-photon imaging: technology developments and biomedical applications. *Chinese Optics Letters* **11**, 011703-011710 (2013).

9. Mertens L, Sonnleitner M, Leach J, Agnew M, Padgett MJ. Image reconstruction from photon sparse data. *Scientific Reports* **7**, 42164 (2017).

10. Mevenkamp N, Binev P, Dahmen W, Voyles PM, Yankovich AB, Berkels B. Poisson noise removal from high-resolution STEM images based on periodic block matching. *Advanced Structural and Chemical Imaging* **1**, (2015).

11. Thakur KV, Damodare OH, Sapkal AM. Poisson Noise Reducing Bilateral Filter. *Procedia Computer Science* **79**, 861-865 (2016).

12. Sparling C, Ruget A, Kotsina N, Leach J, Townsend D. Artificial Neural Networks for Noise Removal in Data-Sparse Charged Particle Imaging Experiments. *ChemPhysChem* **22**, 76-82 (2021).

13. Sarker IH. Machine Learning: Algorithms, Real-World Applications and Research Directions. *SN Computer Science* **2**, 160 (2021).

14. Holzinger A. Introduction to Machine Learning & Knowledge Extraction (MAKE). *Machine Learning and Knowledge Extraction* **1**, 1-20 (2017).

15. Kubat M. *An Introduction to Machine Learning*, 1st edn. Springer International Publishing : Imprint: Springer, (2015).

16. Choi RY, Coyner AS, Kalpathy-Cramer J, Chiang MF, Campbell JP. Introduction to Machine Learning, Neural Networks, and Deep Learning. *Transl Vis Sci Technol* **9**, 12 (2020).

17. Theobald O. *Machine learning for absolute beginners: a plain English introduction*. Scatterplot press (2018).

18. Bennaceur A, Meinke K. Machine Learning for Software Analysis: Models, Methods, and Applications. Springer International Publishing (2018).

19. Sutton RS, Barto AG. *Reinforcement learning : an introduction*, Second edition. edn. The MIT Press (2018).

20. Chadaga K, Prabhu S, Vivekananda BK, Niranjana S, Umakanth S. Battling COVID-19 using machine learning: A review. *Cogent Engineering* **8**, 1958666 (2021).

21. Aggarwal CC. *Neural Networks and Deep Learning: A Textbook*. Springer, Cham (2018).

22. Lavine BK, Blank TR. *Feed-Forward Neural Networks*. Elsevier Science Bv (2009).

23. Li XaW, Xihong. Constructing Long Short-Term Memory based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition. *arXiv*, (2014).

24. Lizhe Tan JJ. Digital Signal Processing. In: *Digital Signal Processing*. Academic Press (2019).

25. Ballard DH. Modular learning in neural networks. In: *Proceedings of the sixth National conference on Artificial intelligence - Volume 1*). AAAI Press (1987).

26. Wheldon C. Convolution of a Gaussian with an exponential. University of Birmingham (2019). [online] Available at: http://www.np.ph.bham.ac.uk/research_resources/programs/halflife/gauss_exp_conv.pdf [Accessed 2 February 2022]

27. Agarap AF. Deep Learning using Rectified Linear Units (RELU). *arXiv*, (2018).