Modern Machine Learning Algorithms: Applications in Nuclear Physics

by

Robert Solli

Thesis

for the degree of

Master of Science

Faculty of Mathematics and Natural Sciences
University of Oslo

April 20, 2019

Contents

List of Figures

Abstract

In this thesis a novel filtering thechinque of AT-TPC noise events is presented using clustering techniques on the latent space produced by a Variational Autoencoder(VAE)

Chapter 1

Introduction

Part I

Machine Learning Theory and Experimental Background

Chapter 2

Machine Learning

2.1 Introduction

The hypothesis explored in this thesis is that we can extract compressed information about physical events from the AT-TPC experiment using modern machine learning methods. With the underlying research question asking what information are we able to capture about nuclear events in the AT-TPC without needing hand-labeling of data. To achieve this we employ the DRAW algorithm (Gregor et al. (2015)). The DRAW algorithm is built of neural network components in a joint architecture comprised of a variational autoencoder wrapped in a long term short term memory cell. Each of the components are discussed in their own sections starting with the neural network in section 2.7 then followed by autoencoders in section 2.9 and finally recurrent neural networks in 2.8.

To arrive at the DRAW network we need to introduce the optimization of the log likelihood function using a binary cross-entropy cost function. In it's simplest form this optimization problem occurs in the formulation of the logistic regression mode introduced in section 2.4. As part of the derivation of the variational autoencoder cost the same optimization problem of the log likelihood will be applied. Likewise we introduce gradient descent methods and regularization, crucial components of modern machine learning, in the familiar framework of linear regression in section 2.5.

Furthermore we hypothesize that this compressed information can be used to linearly separate events in classes. The linear separation of data serves as a precursor to using clustering algorithms to entirely remove the need of researcher intervention in labeling AT-TPC data. In the experiment at hand we hope to separate proton and carbon elastic scattering in a magnetic field. The strength of this model is that it does not rely on hand-labeled data. We demonstrate this first by applying the DRAW algorithm to simulated data showing that we can construct very good separations before turning to real data.

2.2 Model Fitting

1. Describe the fitting of functions to data
2. Overfitting and underfitting and measures to prevent both

2.3 Gradient Descent

1. Introduce gradient descent and its simplest parameter
2. Describe gradient descent variations

2.4 Logistic Regression

2.5 Linear Regression

2.6 Hyperparameters

1. Describe hyperparameter searches and their objective and impact on performance.

2.7 Neural networks

While the basis for the modern neural network was laid more than a hundred years ago in the late 1800's what we think of as neural networks in modern terms was proposed by McCulloch and Pitts (1943). They described a computational structure analogous to a human neuron. Dubbed an Artificial Neural Network (ANN) it takes input from multiple sources, weights that input and produces an output if the signal from the weighted input is strong enough. A proper derivation will follow but for the moment we explore this simple intuition. These artificial neurons are ordered in layers, each successively passing information forward to a final output. The output can be categorical or real-valued in nature. A simple illustration of two neurons in one layer is provided in figure 2.1
The ANN produces an output by a "forward pass". If we let the input to an ANN be $x \in \mathbb{R}^N$, and letting the matrix $W \in \mathbb{R}^{N \times D}$ be a representation of the weight matrix forming the connections between the input and the artificial neurons. Lastly we define the activation function $a(x)$ as a monotonic, once differentiable, function on $\mathbb{R}^1$. The function $a(x)$ determines the complexity of the neural network together with the number of neurons per layer and number of layers. For any complex task the activation takes a non-linear form which

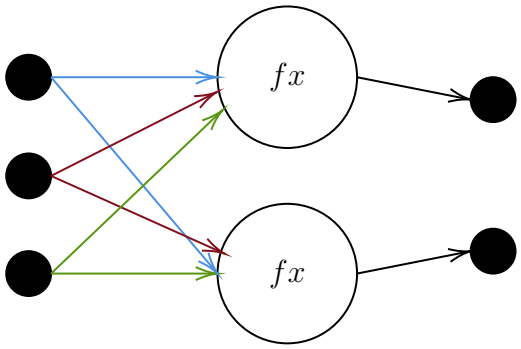

Figure 2.1: An illustration of the graph constructed by two artificial neurons with three input nodes. Colored lines illustrate that each of the input nodes are connected to each of the neurons in a manner we denote as fully-connected.

allows for the representation of more complex problems. A layer in a network implements what we will call a forward pass as defined in function 2.1.

$$\hat{y} = a(\langle x|W\rangle)_D \tag{2.1}$$

In equation 2.1 the subscript denotes that the function is applied element-wise and we denote the matrix inner product in bra-ket notation with $\langle\cdot|\cdot\rangle$. Each node is additionally associated with a bias node ensuring that even zeroed-neurons can encode information. Let the bias for the layer be given as $b \in \mathbb{R}^D$ in keeping with the notation above. Equation 2.1 then becomes:

$$\hat{y} = a(\langle x|W\rangle)_D + b \tag{2.2}$$

As a tie to more traditional methods we note that if we only have one layer and a linear activation $a(x) = x$ the ANN becomes the formulation for a linear regression model. In our model the variables that need to be fit are the elements of W that we denote W_{ij}. While one ordinarily solves optimization problem for the linear regression model by matrix inversion, we re-frame the problem in more general terms here to prime the discussion of the optimization of multiple layers and a non linear activation function. The objective of the ANN is formulated in a "loss function", which encodes the difference between the intended and achieved output. The loss will be denoted as $\mathcal{L}(y, \hat{y}, W)$. Based on whether the output is described by real values, or a set of probabilities this function, $\mathcal{L}$, takes on the familiar form of the Mean Squared Error or in the event that we want to estimate the likelihood of the output under the data; the binary cross-entropy. We will also explore these functions in some detail later. The ansatz for our optimization procedure is given in the well known form of a gradient descent procedure in equation 2.3

$$W_{ij} \leftarrow -\eta \frac{\partial \mathcal{L}}{\partial W_{ij}} + W_{ij} \tag{2.3}$$

2.7.1 Backpropagation

In the vernacular of the machine learning literature the aim of the optimization procedure is to "train" the model to perform better on the regression, reconstruction or classification task at hand. Training the model requires the computation of the total derivative in equation 2.3. This is also where the biological metaphor breaks down, as the brain is almost certainly not employing an algorithm so crude as to be formulated by gradient descent. Backpropagation, or automatic differentiation, first described by Linnainmaa (1976), is a method of computing the partial derivatives required to go from the gradient of the loss w.r.t the output of the ANN to the gradient w.r.t the individual neuron weights in the layers of the ANN. The algorithm begins with computing the total loss, here exemplified with the squared error function, in equation 2.4

$$E = \mathcal{L}(y, \hat{y}, W) = \frac{1}{2} \sum_{n} \sum_{j} (y_{nj} - \hat{y}_{nj})^2 \tag{2.4}$$

The factor one half is included for practical reasons to cancel the exponent under differentiation. As the gradient is multiplied by an arbitrary learning rate η this is ineffectual on the training itself. The sums define an iteration over the number of samples, and number of output dimensions respectively. Taking the derivative of 2.4 w.r.t the output, $\hat{y}$, we get

$$\frac{\partial E}{\partial \hat{y}_j} = \hat{y}_j^M - y_j \tag{2.5}$$

Recall now that for an ANN with M layers the output fed to the activation function is

$$x_j^M = \langle a^{M-1} | W^M \rangle + b_j \tag{2.6}$$

Where the superscript in the inner product denote the output of the second-to-last layer and the weight matrix being the last in the layers. The vector x_j is then fed to the activation to compute the output

$$\hat{y}_j^M = a(x_j^M) \tag{2.7}$$

The activation function was classically the sigmoid (logistic) function but during the last decade the machine learning community has shifted to largely using the rectified linear unit (ReLU) as activation. Especially after the success of Krizhevsky et al. (2012) with AlexNET in image classification. Depending on the output (be it regression or classification) it might be useful to apply the identity transform or a soft max function in the last layer. This does not change the derivation except to change the derivatives in the last layer. We here exemplify the back propagation with the ReLU, which has the form

$$\text{ReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.8}$$

The ReLU is obviously monotonic and its derivative can be approximated with the Heaviside step-function.

$$H(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.9}$$

We again make explicit that the choice of equations 2.4, 2.8 and 2.9 is not a be-all-end-all solution but chosen for their ubiquitous nature in modern machine learning. We then return to equation 2.5 and manipulate the expression via the chain rule

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial x_j} \tag{2.10}$$

The second derivative of the r.h.s we know from our choice of the activation to be equation 2.9, inserting to evaluate the expression we find

$$\frac{\partial E}{\partial x_j^M} = (\hat{y}_j - y_j) H(x_j^M) \tag{2.11}$$

To complete the derivation we further apply the chain rule to find the derivative in terms of the weight matrix elements.

$$\frac{\partial E}{\partial w_{ij}^M} = \frac{\partial E}{\partial x_j^M} \frac{\partial x_j^M}{\partial w_{ij}^M} \tag{2.12}$$

Recall the definition of x_j as the affine transformation defined in equation 2.1. The derivative of the inner product w.r.t the matrix elements is simply the previous layers output. Inserting this derivative of equation 2.6 we have the expression for our derivatives of interest.

$$\frac{\partial E}{\partial w_{ij}} = (\hat{y}_j - y_j) H(x_j) \text{ReLU}(x_i^{M-1}) \tag{2.13}$$

Separately we compute the derivatives of 2.11 in terms of the bias nodes.

$$\frac{\partial E}{\partial b_j} = \frac{\partial E}{\partial x_j} \frac{\partial x_j}{\partial b_j} = (\hat{y}_j - y_j) H(x_j) \cdot 1 \tag{2.14}$$

This procedure is then repeated for the earlier layers computing the $\partial E / \partial w$ as we go. The backward propagation framework is highly generalizable to variations of activation functions and network architectures. The two major advancements

in the theory of ANNs are both predicated on being fully trainable by the back-propagation of errors. Before we consider these improvements made by the introduction of recurrent neural networks (RNN) and convolutional neural networks (CNN) we remark that not only are we free to chose the activation function remarkably freely the backpropagation algorithm also makes no assumptions on the transformation that constructs x_j. As long as it is once differentiable in terms of w_{ij} we are free to pick this transformation also.

there should be a note on the importance of initialization of the weights

2.8 Recurrent Neural Networks

2.8.1 Introduction to recurrent neural networks

The recurrent neural network (RNN) models a unit that has "memory". The memory is encoded as a state variable which is ordinarily concatenated with the input when the network predicts. The model predictions enact a sequence which has led to applications in the generation of text, time series predictions and other serialized applications. RNNs were first discussed in a theoretical paper by Jordan, MI in 86' but implemented in the modern temporal sense by Pearlmutter (1989). A simple graphical representation of the RNN cell is presented in figure 2.2

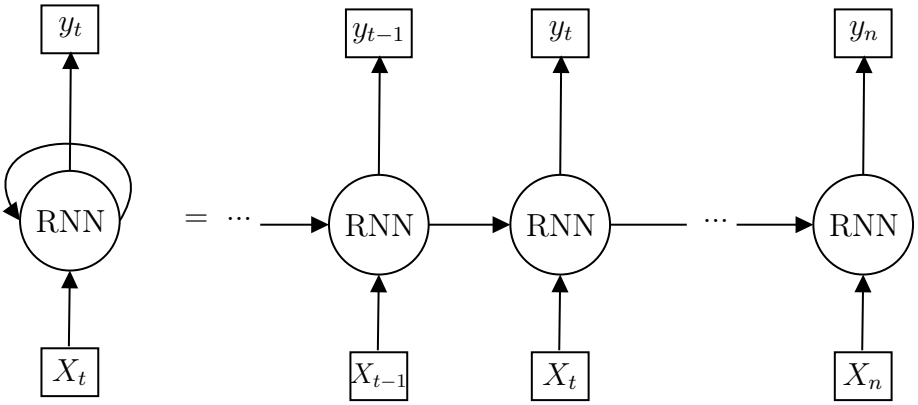

Figure 2.2: A graphical illustration of the RNN cell. The self-connected edge in the left hand side denotes the temporal nature we unroll on the right side. The cell takes as input a state vector and an input vector at time t, and outputs a prediction and the new state vector used for the next prediction. Internally the simplest form this operation takes is to concatenate the state vector with the input and use an ordinary dense network as described in section 2.7 trained with back-propagation.

2.9 Autoencoders

2.9.1 Introduction to autoencoders

An Autoencoder is an attempt at learning a distribution over some data by reconstruction. The interesting part of the algorithm is in many applications that it is in the family of latent variable models. Which is to say the model encodes the data into a lower dimensional latent space before reconstruction. The goal, of course, is to learn the distribution $P(\mathcal{X})$ over the data with some parametrized model $Q(\mathcal{X}|\theta)$ The model consists of two discrete parts ; an encoder and a decoder. Where the encoder is in general a non linear map ψ.

$$\psi : \mathcal{X} \to \mathcal{Z}$$

Where $\mathcal{X}$ and $\mathcal{Z}$ are arbitrary vector spaces with $\dim(\mathcal{X}) > \dim(\mathcal{Z})$. The second part of the model is the decoder that maps back to the original space.

$$\phi : \mathcal{Z} \to \mathcal{X}$$

The objective is then to find the configuration of the two maps ϕ and ψ that gives the best possible reconstruction, i.e the objective $\mathcal{O}$ is given as

$$\mathcal{O} = \arg\min_{\phi,\psi} ||X - \phi \circ \psi||^2 \tag{2.15}$$

Where the $\circ$ operator denotes function composition in the standard manner. As the name implies the encoder creates a lower-dimensional "encoded" representation of the input. This objective function is optimized by a mean-squared-error cost in the event of real valued data, but more commonly through a binary crossentropy for data normalized to the range $[0, 1]$. This representation can be useful for identifying whatever information-carrying variations are present in the data. This can be thought of as an analogue to Principal Component Analysis (PCA) (Marsland (2009)). In practice the non-linear maps, ψ and ϕ, are most often parametrized and optimized as ANNs. ANNs are described in detail in section 2.7. The autoencoder was used perhaps most successfully in a denoising tasks. More recently the Machine Learning community discovered that the decoder part of the network could be used for generating new samples form the sample distribution, dubbed "Variational Autoencoders" they are among the most useful generative algorithms in modern machine learning.

Citation needed. Also should I include example of denoising autoencoders ? Maybe a description at least.. Link to notebook maybe?

2.9.2 Variational Autoencoder

Originally presented by Kingma and Welling (2013) the Variational Autoencoder (VAE) is a twist upon the traditional autoencoder. Where the applications of an ordinary autoencoder largely extended to de-noising with some authors using it for dimensionality reduction before training an ANN on the output the VAE seeks to control the latent space of the model. The goal is to be able to generate samples from the unknown distribution over the data. Imagine trying to draw a sample from the distribution of houses, we'd be hard pressed to produce anything remotely useful but this is the goal of the VAE. In this thesis the generative properties of the algorithm is only interesting as a way of describing the latent space. Our efforts largely concentrate on the latent space itself and importantly discerning whether class membership, be it a physical property or something more abstract [1]xw is encoded.

The variational autoencoder cost

In section 2.9.1 we presented the structure of the autoencoder rather loosely. For the VAE which is a more integral part of the technology used in the thesis a more rigorous approach is warranted. We will here derive the loss function for the VAE in such a way that makes clear how we aim to impose known structure of the latent space. We begin by considering the family of problems encountered in variational inference, where the VAE takes its theoretical inspiration from. We define the joint probability distribution of some hidden variables z and our data x conditional on some β. In a traditional modeling context we would coin z as including model parameters and β would then denote the hyperparameters. The variational problem is phrased in terms of finding the posterior over z, given β

$$p(z|x,\beta) = \frac{p(z,x|\beta)}{\int_z p(z,x|\beta)} \tag{2.16}$$

citation?

The integral in the denominator is intractable for most interesting problems . This is also the same problem that Markov Chain Monte Carlo (MCMC) methods aim at solving. In physics this family of algorithms has been applied to solve many-body problems in quantum mechanics primarily by gradient descent on variational parameters .

citation? Comph-phys 2 compendium?

Next we introduce the Kullback-Leibler divergence (KL-divergence) (Kullback and Leibler (1951)) which is a measure of how much two distributions are alike, it is important to not that it is however not a metric. We define the KL-divergence in equation 2.17 from a probability measure P, to another Q, by their probability density functions p, q over the set $x \in \mathcal{X}$.

[1] examples include discerning whether a particle is a proton or electron, or capturing the "five-ness" of a number in the MNIST dataset

$$D_{KL}(P||Q) = -\int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx \tag{2.17}$$

$$= \langle \log\left(\frac{p(x)}{q(x)}\right) \rangle_p \tag{2.18}$$

In the context of the VAE the KL-divergence is a measure of dissimilarity of P approximating Q (Burnham et al. (2002)). The derivation then sensibly starts with a KL-divergence.

We begin by defining $q(z|x)$ to be the true posterior distribution over the latent variable $z \in \mathcal{Z}$, conditional on our data $x \in \mathcal{X}$ with a true posterior distribution $p(x)$ and $q(z)$, with an associated probability measure Q as per our notation above. Let then the distribution over the latent space parametrized by the autoencoder be given as $\psi(z|x)$, where the autoencoder parametrizes a distribution $\eta(x)$, and an associated probability measure Ψ. And recalling Bayes rule for conditional probability distributions $p(z|x) = (p(x|z)p(z)/p(x)$ we then have

$$D_{KL}(\Psi||Q) = \langle \log\left(\frac{\psi(z|x)}{q(z|x)}\right) \rangle_\psi \tag{2.19}$$

$$= \langle \log\left(\psi(z|x)\right) \rangle_\psi - \langle \log\left(p(x|z)q(z)\right) \rangle_\psi + \log\left(p(x)\right) \tag{2.20}$$

$$= \langle \log\left(\frac{\psi(z|x)}{q(z)}\right) \rangle_\psi - \langle \log\left(p(x|z)\right) \rangle_\psi + \log\left(p(x)\right) \tag{2.21}$$

Note that the term $-\langle \log\left(p(x|z)\right) \rangle_\psi$ is the log likelihood of our decoder network which we can optimize with the cross entropy as discussed in section 2.4. Rearranging the terms we arrive at the variational autoencoder cost

$$\log(p(x)) - D_{KL}(\Psi||Q) = \langle \log\left(p(x|z)\right) \rangle_\psi - \langle \log\left(\frac{\psi(z|x)}{q(z)}\right) \rangle_\psi \tag{2.22}$$

We are still bound by the intractable integral defining the evidence $p(x) = \int_z p(x,z)$ which is the same integral as in the denominator in equation 2.16. The solution appears by approximating the KL-divergence up to an additive constant by estimating the evidence lower bound (ELBO). This function is defined as

$$ELBO(q) = \langle \log(p(z,x)) \rangle - \langle \log(q(z|x)) \rangle \tag{2.23}$$

To fit the VAE cost we rewrite the ELBO in terms of the conditional distribution of x given z

$$ELBO(q) = \langle \log(p(z)) \rangle + \langle \log(p(x|z)) \rangle - \langle \log(q(z|x)) \rangle \tag{2.24}$$

Finally the ELBO can be related to the VAE loss by applying Jensen's inequality (J) to the log evidence

$$\begin{aligned}
\log(p(x)) &= \log \int_z p(x|z)p(z) && (2.25)\\
&= log \int_z p(x|z)p(z)\frac{q(z|x)}{q(z|x)} && (2.26)\\
&= log\langle p(x|z)p(z)/q(z|x)\rangle && (2.27)\\
&\overset{(J)}{\geq} \langle \log(p(x|z)p(z)/q(z|x))\rangle && (2.28)\\
&\geq \langle \log(p(x|z))\rangle + \langle \log(p(z))\rangle - \langle \log(q(z|x))\rangle && (2.29)
\end{aligned}$$

Which shows that the function enacted by the ELBO approximates the VAE loss up to a constant i.e. the KL loss on the RHS in equation 2.22. Kingma and Welling (2013) showed that this variational lower bound on the marginal likelihood of our data is feasibly approximated with a neural network when trained with backpropagation and gradient descent methods. That is we estimate the derivative of the ELBO with respect to the neural network parameters, as described by the backpropagation algorithm in section 2.7.1. We note again that in the above notation we would parametrize the distribution $p(x|z)$ as a neural network, in machine learning parlance called the generator network and denoted as ϕ in section 2.9.

2.10 TODO

1. Add section on preceptron for a simple explanation of the backpropagation algorithm.
2. Write out section on RNN
3. Write out section on LSTM
4. Write out section on DRAW
5. Write out section on convolutions

Chapter 3

Experimental background

3.1 Introduction

1. Describe the FRIB facility and its goals
2. Describe the advent and use of ML in High Energy particle physics
3. Contextualize the need for ML in Nuclear physics, what has changed?

3.2 Active Target Time Projection Chambers

1. Introduce the TPC and AT-TPC
2. Introduce the objective and of the AT-TPC experiments
3. Introduce and discuss the physics of the AT-TPC experiments
4. Discuss the need for ML in event categorization in the AT-TPC experiments

Part II

Implementation

Chapter 4

Methods

4.1 Introduction

In this chapter we will illustrate the research pipeline applied to the experimental data from the AT-TPC. We will show the implementation of the algorithms described in section 2, and their performance on simulated data to illustrate their workings and to establish a baseline for the inquiry into the real experimental data.

The implementation of the algorithms described in chapter 2 have been implemented for this thesis in the python programming language (van Rossum and Python development team (2018)). Parts of algorithms displayed for demonstration or exposition have been developed in the numerical python framework numpy (van der Walt et al. (2011)). While variational autoencoder and DRAW algorithms were implemented in the tensorflow framework (Abadi et al. (2015)). The choice of python as the framework in which to develop this thesis was made for the ease of rapid prototyping and the extensive availability of mature numerical libraries like the ones cited above. Plots of numerical performance and data visualization was achieved with the matplotlib graphics package for python (Hunter (2007)).

We begin by describing the tensorflow framework as it is in that context the algorithms are implemented

4.2 Tensorflow

The numerical framework tensorflow is a development for deep learning tasks developed by Google brain starting in 2011.

Part III

Results

Part IV

Discussion and Conclusion

Chapter 5

Notes

1. L1 regularization on the LSTM cells in the draw network seem to encourage the network to capture "many events". Looks like many spirals in one. While L2 (or sparese) regularization represents the images well. Can we represent the inner workings of the LSTM in some way?
2. Benchmark reconstruction loss for DRAW is at 255 - 1200 nodes, 60 filters, 10 timesteps, L2 regularization, Adam optimizer
3. Nesterov momentum yields suboptimal results. Reconstruction loss of about 1.4 times the loss when using Adam
4. Adadelta yields pure noise reconstructions (short simulation)
5. Adagrad yields localized "clouds" in the output
6. for simulated data it seems we can compress to about $350 \sim 300$ nodes in the encoder lstm. And to 3 dimensions in the latent space
7. In what seems like the minimal compressed state for the simulated data the training seems unstable and will frequently get stuck in local minima or have the gradient explode
8. DRAW without attention seems unable to learn even the simulated distribution at 128 by 128 pixels
9. In the DRAW algorithm the glimpse is specified by an affine weight transformation - but to be comparable it should be constant as a hyperparameter.
10. Implementing the glimpse as a hyperparameter was hugely successful, perhaps surprisingly in decreasing the reconstruction loss. Now remains the task of using the latent representations for classification
11. Two class-classification on the latent space was also hugely successful for simulated data

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., and Research, G. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. Technical report.

Burnham, K. P., Anderson, D. R., and Burnham, K. P. (2002). *Model selection and multimodel inference : a practical information-theoretic approach*. Springer.

Gregor, K., Com, D., Rezende, D. J., and Wierstra, D. (2015). DRAW: A Recurrent Neural Network For Image Generation Ivo Danihelka. *Proceedings of Machine Learning Research*, 37.

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3):90–95.

Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks.

Kullback, S. and Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.

Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. *BIT*, 16(2):146–160.

Marsland, S. (2009). Machine Learning: An Algorithmic Perspective.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

Pearlmutter, B. A. (1989). Learning State Space Trajectories in Recurrent Neural Networks. *Neural Computation*, 1(2):263–269.

van der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering*, 13(2):22–30.

van Rossum, G. and Python development team, T. (2018). Python Tutorial Release 3.7.0 Guido van Rossum and the Python development team. Technical report.