Encoder-decoder neural network for solving the nonlinear Fokker-Planck-Landau collision operator in XGC

M.A. Miller[1,3]†, R.M. Churchill[1], A. Dener[2], C.S. Chang [1], T. Munson [2], R. Hager [1]

[1]Plasma Physics Laboratory, 100 Stellarator Road, Princeton, NJ 08540, USA

[2]Argonne National Laboratory, 9700 S Cass Ave, Lemont, IL 60439, USA

[3]Columbia University, Applied Physics and Applied Mathematics Department, New York, NY 10027, USA

An encoder-decoder neural network has been used to examine the possibility for acceleration of a partial integro-differential equation, the Fokker-Planck-Landau collision operator. This is part of the governing equation in the massively parallel particle-in-cell code, XGC, which is used to study turbulence in fusion energy devices. The neural network emphasizes physics-inspired learning, where it is taught to respect physical conservation constraints of the collision operator by including them in the training loss, along with the ℓ_2 loss. In particular, network architectures used for the computer vision task of semantic segmentation have been used for training. A penalization method is used to enforce the "soft" constraints of the system and integrate error in the conservation properties into the loss function. During training, quantities representing the density, momentum, and energy for all species of the system is calculated at each configuration vertex, mirroring the procedure in XGC. This simple training has produced a median relative loss, across configuration space, on the order of 10^{-4}, which is low enough if the error is of random nature, but not if it is of drift nature in timesteps. The run time for the current Picard iterative solver of the operator is $O(n^2)$, where n is the number of plasma species. As the XGC1 code begins to attack problems including a larger number of species, the collision operator will become expensive computationally, making the neural network solver even more important, especially since the training only scales as $O(n)$. A wide enough range of collisionality has been considered in the training data to ensure the full domain of collision physics is captured. An advanced technique to decrease the losses further will be discussed, which will be subject of a subsequent report. Eventual work will include expansion of the network to include multiple plasma species.

1. Background

As high performance computing (HPC) initiatives bring the advent of the exascale era, numerical simulations become capable of providing prediction with high levels of accuracy. In toroidal magnetic confinement fusion devices, the hot plasma dynamics is determined by space-time overlapping multi-scale interactions among complex multiscale physics. Calculating these interactions is computationally expensive, especially in the case of total-f gyrokinetic codes, which solve for the motion of particles on turbulence scales and the background scale together without scale separation. XGC (X-point Gyrokinetic Code) is a massively parallel hybrid Lagrangian-Eulerian particle-in-

---

† Email address for correspondence: mam2510@columbia.edu

cell based gyrokinetic code focused on simulating the highly nonlinear, non-equilibrium edge region of fusion devices (Ku *et al.* 2018). An expensive part of the code is the collision operator acting on the five-dimensional (5D) particle distribution functions. XGC uses the nonlinear Fokker-Planck-Landau (FPL) collision operator (Landau 1936), seen in Equation 1.1, on a two-dimensional normalized velocity grid, $v_\perp/v_{th}, v_\parallel/v_{th}$ (spatial points are independent in the collision operator) (Yoon & Chang 2014; Hager *et al.* 2016). At every time step, particles tracked in 3D configuration space and 2D velocity coordinates are histogrammed into velocity particle distribution functions f, on regular, 2D velocity grids at each configuration space vertex. An implicit Picard iteration scheme is then used to solve the integro-differential equation involving the collision operator on the 2D velocity grid at each configuration space vertex, with a typical number of configuration space vertices in a simulation on the order of 10^1 – 10^2 million.

$$\frac{df_a}{dt} = \sum_b C_{ab}(f_a; f_b') = -\sum_b \frac{e_a^2 e_b^2 ln\Lambda_{ab}}{8\pi\epsilon_0^2 m_a} \nabla_v \cdot \int \mathbf{U} \cdot \left(\frac{f_a}{m_b}\nabla_v' f_b' - \frac{f_b'}{m_a}\nabla_v f_a\right) d^3v' \quad (1.1)$$

In Equation 1.1, a and b denotes separate species, although self-collisions are included in the case that $b = a$. f_a and f_b are the particle distribution functions of the species, $e_{a,b}$ is the charge, $m_{a,b}$ is the mass, $\ln \Lambda_{ab}$ is the Coulomb logarithm, and $\mathbf{U}$ is a tensor that is a function of the relative vector $\mathbf{u} = \mathbf{v} - \mathbf{v}'$:

$$\mathbf{U} = \frac{u^2\mathbf{I} - \mathbf{uu}}{u^3} \quad (1.2)$$

The run time for the current Picard iteration solver of the collision operator is $O(n^2)$, where n is the number of plasma species. As the XGC1 code begins to attack problems including a larger number of species, such as including impurities like tungsten and its multiple charge states in devices like ITER, the collision operator will become the dominant computation (Dominski *et al.* 2019).

Deep neural networks have proven successful for many computer vision tasks, including image recognition, as well as text and speech recognition. Given their success, a great deal of effort has been expended in utilizing them to solve complex problems across science and engineering. Specifically in physics, their application is widespread. From learning field theories using a set of training data produced from a physical field (Qin 2019), to simulating particles using a graph network by combining an ODE integrator with Hamiltonian dynamics (Sanchez-Gonzalez *et al.* 2019), to predicting non-linear, cosmological large scale structures of the universe (He *et al.* 2019), neural networks have proven incredibly versatile.

Recognizing the importance of efficient PDE solvers across the field of physics, a more concerted effort is underway to use neural networks to help solve especially intractable PDEs. Almost always, training of a PDE solver using neural networks depends on minimizing an objective function that is composed primarily of the quadratic residual, also known as mean-squared error or ℓ_2 loss. Consider a stationary PDE of the form:

$$\begin{aligned} Lu = f, x \in \Omega \\ Bu = g, x \in \Gamma \subset \partial\Omega \end{aligned} \quad (1.3)$$

where L is a differential operator, f is a forcing function, B is a boundary operator, and g is boundary data. Solutions then require the minimization of an objective function composed of the ℓ_2 loss as follows:

$$C = \alpha||L\hat{u} - f||^2 \quad (1.4)$$

where $\hat{u} = \hat{u}(\theta, \phi, ...)$ is the ansatz that depends on some number of trainable parameters θ, ϕ, etc. that can be optimized to minimize Equation 1.4. Here α is some coefficient that can vary and that becomes especially important as other terms are added to the objective function.

While this methodology is common to PDE neural network techniques, the incorporation of boundary conditions, or any constraints in general varies. Often, these constraints are built into the network itself as "hard" constraints, either directly into the architecture or into a constraint on the ansatz. Some attempt to do the former by adding a differentiable PDE layer that enforces spatial constraints (Jiang *et al.* 2020) or in the case of a simple governing PDE, the use of an untrainable "physics embedded decoder" that replicates the differential operator and boundary conditions directly (Arvind T. Mohan *et al.* 2017). Others use the boundary conditions to constrain the ansatz itself and then proceed with unconstrained optimization of an objective function like in Equation 1.4 (Berg & Nyström 2018). One last set of techniques treats the constraints as "soft" and instead incorporates them into the objective function as penalty terms. This is done by adding the boundary conditions directly into the loss function (Sirignano & Spiliopoulos 2018) or by leveraging the use of a coefficient like α in Equation 1.4 that allows for a relative prioritization of different constraints in the loss function (Beucler *et al.* 2019).

This paper employs a technique very similar to the latter. Using a penalization method to optimize "soft" constraints, it aims to learn the nonlinear transformation of the collision operator in Equation 1.1, enabling the XGC code to be accelerated. Since the FPL collision operator can be expressed as solving the collisional change $\Delta f_a = F(f_a, f_b, ...)$, where Δf_a can ordered as the same size on the normalized 2D velocity grid as f_a, f_b, etc. in a non-equilibrium edge plasma, here we use encoder-decoder deep neural networks used in the common computer vision task of semantic segmentation.

2. Machine Learning in Computer Vision

Computer vision has been one of the largest drivers for machine learning algorithm development. During the ImageNet competition, it was quickly noticed that deep convolutional neural networks (CNN) performed best for image recognition (Alom *et al.* 2017). These networks take in an input image and outputs a set of scores corresponding to a set of possible categories that identify the image. The largest of these scores corresponds to the prediction of the network. Structurally, the input has dimensions H x W x 3, where H is the number of image pixels vertically, W is the same horizontally, and 3 corresponds to the three color channels, red, green, and blue. The output is simply a 1D vector with C elements, where C is the prescribed number of possible identification classes. A CNN contains a variable number of hidden layers which perform convolutions on the input image in an attempt to discern underlying features to aid in identification. At each of these layers, a convolution kernel performs a convolution, or formally a cross-correlation, on the data. For neural networks, this is performed in 2D as follows (although the operation generalizes to higher dimensions):

$$(f * g)(i, j) = \sum_{m=0}^{M} \sum_{n=0}^{N} f(m, n)g(i + m, j + n) \tag{2.1}$$

where f is the input into the convolution layer, g. For a function represented by a matrix, as is the case for neural networks, m and n simply iterate over the two dimensions of the matrix, M and N, respectively (Kao 2017).

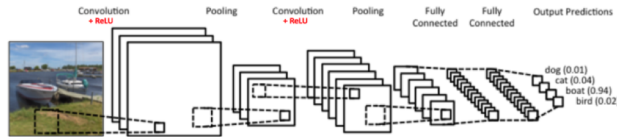Along with these convolutional layers, the data is passed through a nonlinear activation

Figure 1: Example schematic of basic convolutional neural network for image recognition. Reproduced from (Karn 2016).
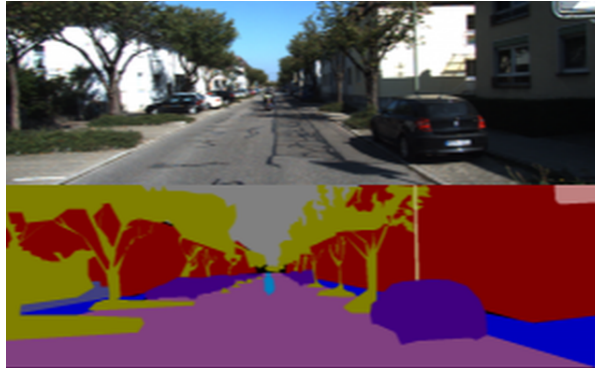

Figure 2: Example image of semantic segmentation being used to identify all objects in an image. Each color in the bottom image corresponds to a different class in the input image, as predicted by the neural network (e.g. yellow for tree, pink for street, etc.). Reproduced from (CVC 2016).

function at each layer, allowing the network the flexibility to learn the nonlinearities associated with many real-world problems. Every few convolutions, the data is passed through a learned filter that "summarizes" the features in the input image. These layers, called pooling layers, reduce the number of parameters in the network. Finally, the CNN ends with a set of fully connected (FC) layers that terminate with a linear operation that transforms the final 2D feature map into a vector containing the C class scores. As with any gradient-based optimization problem, a loss function allows for computation of gradients and updating of the weights of the network to attempt to minimize the loss (Kao 2017). The full process, including pooling layers, is illustrated in Figure 1.

As computer vision networks increased in sophistication, they turned to more complicated tasks like that of semantic segmentation. This involves classifying each pixel in an image, allowing the network to identify every part of an image. An example of an image parsed by a neural network with this goal is shown in Figure 2. This framework of making pixel-wise decisions on a input dataset is of great use to a variety of unrelated problems, in particular to making accelerating solutions to the equations governing the collision of particles in a tokamak. To understand why, it is necessary once again to consider the spatial arrangement of data that semantic segmentation techniques manipulate.

Semantic segmentation is the computer vision task of labeling each pixel with the object class that pixel belongs to, e.g. marking all pixels that belong to a car in an image. For semantic segmentation, the input is an image of dimensions H x W x 3. As each pixel receives a class score, the output takes the form H x W x C. Since the image has H x W pixels, the output array must contain H x W vectors of C elements. The decision of what a certain object is now made at a pixel level, H x W times. For each of these pixels, there is an associated cross-entropy loss function from which the total

overall loss is computed, which is then used in backpropagation to update the overall gradients.

3. Computer Vision Technique for Collision Operator

Semantic segmentation techniques are a good starting point for attacking the collision operator prediction problem, specifically because of the similarities between the hierarchies of convolutions in CNNs and the velocity-space gradients in the FPL collision operator. To perform the collision calculation, at every timestep, XGC performs particle to mesh interpolation, creating distribution functions on regular 2D velocity grids of size $N_{v_{perp}}$ x $N_{v_{par}}$ at each configuration grid vertex. This array can be thought of as a set of bins representing the number of particles at particular velocities, both parallel and perpendicular to the magnetic field lines.

XGC creates distribution functions on each mesh vertex for each particle type simulated. In this case, we consider electrons and a single ion species. The collision operator takes in these distribution functions and performs the collision calculation according to the FPL equations and outputs the change in the distribution function of species a, labeled $\Delta f_{a,col}$. For example the ion $\Delta f_{i,col}$ is:

$$\Delta f_{i,col} = \Delta t \cdot C_i(f_i, f_e) = \Delta t \cdot [C_{ii}(f_i, f_i) + C_{ie}(f_i, f_e)], \tag{3.1}$$

where Δt is the collision timestep size. This $\Delta f_{a,col}$ has the same configuration space size as the original distribution function, much like semantic segmentation techniques, which preserve the size of an input image.

One main difference between the collision operator and typical semantic segmentation is $\Delta f_{a,col}$ is continuous, such that it's a regression problem instead of a classification problem. We use a mean-squared error loss (or ℓ_2) as part of the loss function:

$$L_{\ell_2} = \frac{1}{N} \sum_{i=1}^{N} (\Delta f_{XGC} - \Delta f_{ML})^2 \tag{3.2}$$

where N is the number of configuration space vertices, Δf_{XGC} is the result of the typical Picard iteration solver used by XGC, and Δf_{ML} is the predicted Δf from the machine learning neural network.

4. Physics-Inspired Constrained Learning

In a general regression problem, the ℓ_2 loss is sufficient to train a network to minimize the loss of a nonlinear transformation. In this problem, however, we are dealing with a physical system that is constrained by physical laws, namely the laws of mass, momentum, and energy conservation. The ℓ_2 loss does not directly take into consideration the conservation laws of the collision operator and is often insufficient to ensure these physical constraints are satisfied. For two species, a and b, the conservation equations can be summarized as follows (Helander & Sigman 2002):

$$\int \phi_a(\mathbf{v}) C_{ab}(f_a, f_b) \, d^3 v = -\int \phi_b(\mathbf{v}) C_{ba}(f_b, f_a) \, d^3 v \tag{4.1}$$

where $\phi(\mathbf{v}) = \{m, m\mathbf{v}, \frac{1}{2}mv^2\}$ represent mass, momentum, and kinetic energy.

For conservation of mass it can be shown more specifically that:

$$\int C_{ab}(f_a, f_b) \, d^3 v = 0 \tag{4.2}$$

Note also that for self-collisions, $a = b$, Equation 4.1 reduces further:

$$\int \phi_a(\mathbf{v}) C_{aa}(f_a, f'_a) \, d^3v = 0 \tag{4.3}$$

4.1. *XGC Implementation*

Details of the Fokker-Planck-Landau collision operator can be found in (Yoon & Chang 2014) and (Hager *et al.* 2016). We list here mathematically the conservation properties that XGC uses in the code. For now, we will present continuous, analytical equations, and not the detailed numerical implementations (see (Hager *et al.* 2016)). Also, we will retain the full 3D-3V notation from Section I, even though XGC is gyrokinetic (3D-2V), and there are a number of differences, but this will suffice to illustrate the main points. We drop the explicit notation inputting the distribution functions into the collision operator (C_{ab}, as it is redundant with the subscripts of the collision operator. For simplicity we limit ourselves to the two-species case, electrons ("e") and a single ion specieis ("i").

Using the following moment definitions:

$$\begin{array}{rcl} n_a &=& \int f_a d^3v \\ P_a &=& \int m_a \mathbf{v} f_a d^3v \\ E_a &=& \int \frac{1}{2} m_a v^2 f_a d^3v \end{array} \tag{4.4}$$

The conservation properties calculated in XGC are as follows:

$$\begin{array}{rcl} \Delta n_i &=& dt \left[\int C_{ii} \, d^3v + \int C_{ie} \, d^3v \right] = 0 \\ \Delta n_e &=& dt \left[\int C_{ee} \, d^3v + \int C_{ei} \, d^3v \right] = 0 \\ \Delta P_i &=& dt \left[\int m_i \mathbf{v} C_{ii} \, d^3v + \int m_i \mathbf{v} C_{ie} \, d^3v \right] \\ \Delta P_e &=& dt \left[\int m_e \mathbf{v} C_{ee} \, d^3v + \int m_e \mathbf{v} C_{ei} \, d^3v \right] \\ \Delta E_i &=& dt \left[\int \frac{1}{2} m_i v^2 C_{ii} \, d^3v + \int \frac{1}{2} m_i v^2 C_{ie} \, d^3v \right] \\ \Delta E_e &=& dt \left[\int \frac{1}{2} m_e v^2 C_{ee} \, d^3v + \int \frac{1}{2} m_e v^2 C_{ei} \, d^3v \right] \end{array} \tag{4.5}$$

These are combined to form the conservation laws (see Equation 4.1 and 4.5:

$$\begin{array}{rcl} \Delta n_i &=& 0 \\ \Delta n_e &=& 0 \\ \Delta P &=& \Delta P_i + \Delta P_e = 0 \\ \Delta E &=& \Delta E_i + \Delta E_e = 0 \end{array} \tag{4.6}$$

Since these are numerical solutions, the terms in Equation 4.6 will not be exactly 0. The quantities in Equation 4.6 when normalized provide a test to determine how close the numerical solver is to satisfying these physical conservation properties. The following gives the conditions for the normalized quantities to satisfy for the algorithm to be considered converged in XGC's numerical Fokker-Planck solver:

$$\text{converged if} \begin{cases} \left| \dfrac{\Delta n_i}{n_i} \right| & < 10^{-10} \\[2ex] \left| \dfrac{\Delta n_e}{n_e} \right| & < 10^{-10} \\[2ex] \dfrac{|\Delta P_i + \Delta P_e|}{|n_i m_i v_{th,i}| + |n_e m_e v_{th,e}|} & < 10^{-7} \\[2ex] \dfrac{|\Delta E_i + \Delta E_e|}{|E_i + E_e|} & < 10^{-7} \end{cases} \tag{4.7}$$
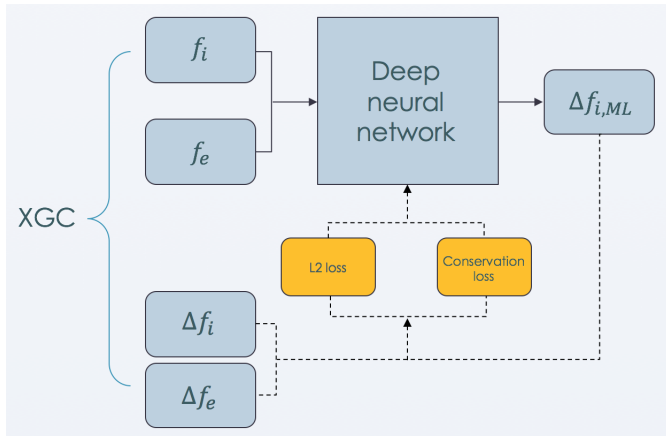
Figure 3: Schematic of the training processes used for collision prediction.

where $v_{th,i}$ and $v_{th,e}$ are local-average ion and electron thermal speeds, $v_{th} = \sqrt{T/m}$. The average electron momentum is usually negligible compared to the ion momentum.

4.2. *ML Implementation*

We enforce these physical conservation constraints in our neural network by including them as regularization terms in the loss:

$$L_{cons} = \sum_{j=1}^{3} \lambda_j \sum_{a=\{i,e\}} \frac{\langle \phi_a^j \, \Delta f_{a,ML} \rangle}{A_j} \tag{4.8}$$

This equation incorporates Equation 4.1 into the loss, seeking to minimize the change in these quantities due to the collision operator. Here, $\phi_a^j = \{m_a, m_a\mathbf{v}, \frac{1}{2}m_a v^2\}$ and $A_j = \{n_{i,e}, |n_i m_i v_{th,i}| + |n_e m_e v_{th,e}|\,, |E_i + E_e|\}$ (as in Equation 14) for ion density = electron density, momentum, and energy conservation, and $\langle \rangle$ represents the numerical velocity integral. The quantity λ_j represents a hyperparameter scaling factor that allows for modification of how heavily each conservation property is weighed in the overall loss function. This in turn affects how much the network corrects for each of the conservation errors compared to the ℓ_2 loss. The overall loss function is simply the sum of L_{ℓ_2}, Equation 3.2 and L_{cons}, Equation 4.8.

5. Training Setup and Architecture

A dataset of the particle distribution functions f_i and f_e were gathered from XGC, and collision kernel output Δf of the existing Picard iteration scheme was generated. These comprised over 2.5 million samples of the $N_{v_\perp}$ x $N_{v_\parallel}$ = 32 x 31 velocity-space grid distribution functions in the present problem. Using the distribution function for both the ions and the electrons allows for calculation of the actual Δf within the XGC collision kernel. The three sets of data, f_i, f_e, and Δf_i can then be used for training the neural network according to the schematic depicted in Figure 3. Δf_e can be included in this framework, and is left for a future publication.

Before training begins, a few steps are taken to preprocess the data. In the interest of reducing the training time, the training is limited to 4 toroidal planes. The data from these planes is filtered further to remove points that have nonphysical f data and that

Figure 4: Schematic of ReSeg, a neural network architecture used for semantic segmentation. Reproduced from (Visin *et al.* 2016).

have target Δf that did not converged when passed into the XGC collision kernel. Once this is done, the entire dataset is divided into three groups: training, validation, and testing, comprising 80%, 10%, and 10% of the data respectively. The training data is then split into equal-sized batches before being passed into the neural network.

Initial training was begun on U-Net (Ronneberger *et al.* 2015), one of the networks that performed best on semantic segmentation techniques earlier on. Once initial results were satisfactory, more sophisticated networks were investigated. In the end, the ReSeg network was chosen. It combines Convolutional (CNN) and Recurrent Neural Networks (RNN) to perform semantic segmentation (Visin *et al.* 2016), and with minor tweaks to the FC linear layer at the end, to replicate the collision operator. ReSeg begins with a CNN, in this case VGG16 (Simonyan & Zisserman 2015). This CNN downsamples the input distribution function, producing a grid that is 256 planes deep. ReSeg then stacks two layers of ReNet, each of which is composed of four RNNs that sweep over the distribution function. Two layers, interspersed with ReLU activations upsample the data to a 32 x 31 x 100 grid. For the purposes of this problem, the last layer of the network includes a convolutional kernel of size 1 x 1 x 100 with stride 1, which outputs the desired 32 x 31 x 1 distribution function representing Δf. For this problem, there are over 2.6 million parameters in the network.

A schematic of the network architecture can be seen in Figure 4. Though it is eventually important to explore a range of devices, initial training was begun on the JET tokamak. Stochastic Gradient Descent (SGD) with momentum was used primarily as the optimizer, though RAdam, a variant of Adam that reduces the variance of adaptive learning rates (Liu *et al.* 2019), also showed success. Training was done for 100 epochs, with a validation set done twice an epoch. The training and validation results are shown in Figure 5, with the ℓ_2 and conservation error shown separately.

6. Initial Training and Validation

Figure 5 shows the training loss over time for 100 epochs of ReSeg training. The learning rate was adjusted using a combination of warmup and manual learning rate decay every 10 epochs. Note that the "Training" and "Validation" plots include the λ_j from Equation 4.8, while the "Conservation" plot does not. It is clear that after around 70 epochs, the loss begins to plateau and learning stagnates. This implores a finer tuning of the hyperparameters, or perhaps a revision of the optimization procedure, specifically in the weights assigned to each quantity in the loss function (λ_j in Equation 4.8). Figure 5b breaks up the conservation loss into each of three conservation properties: ion density, momentum, and energy, the three elements of the summation in Equation 4.8. Finally, Figure 6 illustrates the difference in learning of the structure of the Δf for two different ψ_n, a normalized radial coordinate, which lie in two different regimes of collisionality. The lower the ψ_n, meaning the closer the plasma is to the core, the more Maxwellian the plasma, and so the smaller resulting Δf from the collision operator. Collisions here result in a smaller Δf_{col}, perhaps making the transformation easier for the network to learn.
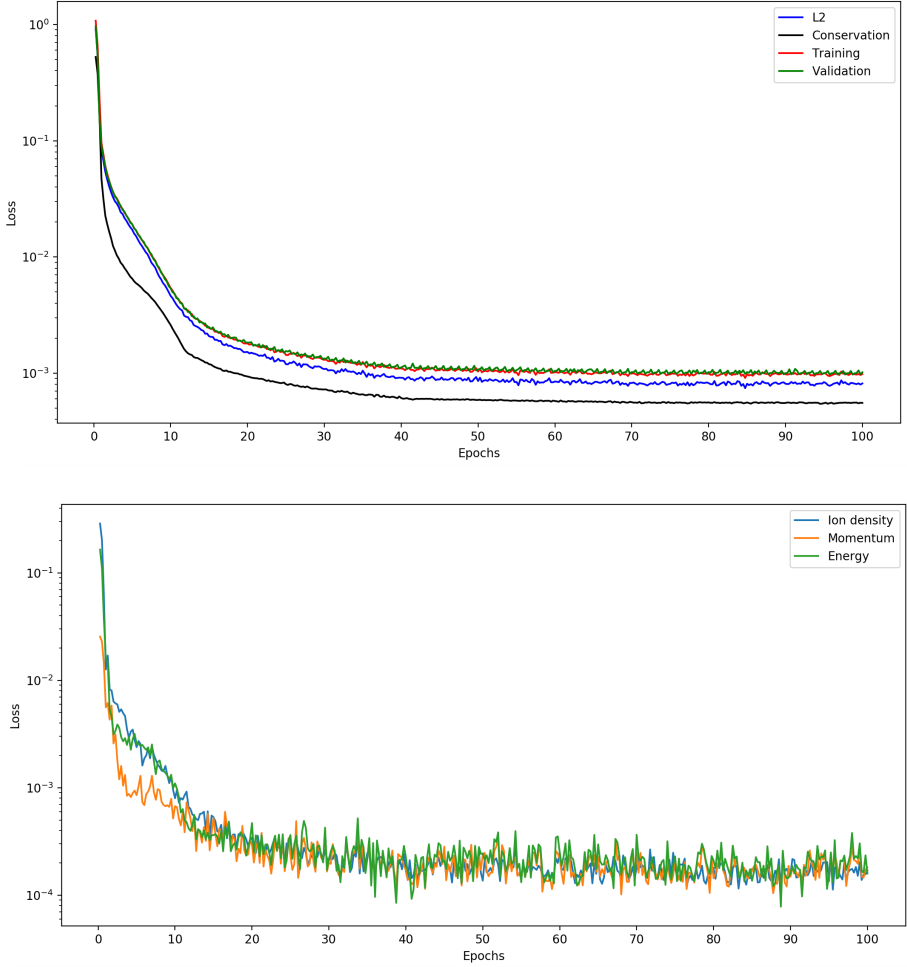
Figure 5: Training results on JET data for 100 epochs and four toroidal planes showing (a) overall training and validation, as well as ℓ_2 and conservation loss, and (b) the three components of the conservation loss separately

In order to declare success, it is necessary to look at how well the neural network can solve Equation 4.1, i.e. how well it conserves mass, momentum, and energy. Looking at Equation 4.8, we require each of the terms in the sum to be a minimum. Work still must be done to reduce the quantities in Figure 7 to the threshold levels in XGC in case there is a drift type ML error propagation in time, but even these results show promise for reaching the precision required for the XGC code. The benefit of course to using the ML algorithm is that the time to solution should be much faster compared to the Picard iteration solver. Further work into optimization schemes has demonstrated even more promising results. In particular, an Augmented Lagrangian scheme, which will be detailed in a subsequent report, has extended this initial ML application down to 10^{-6} accuracy.
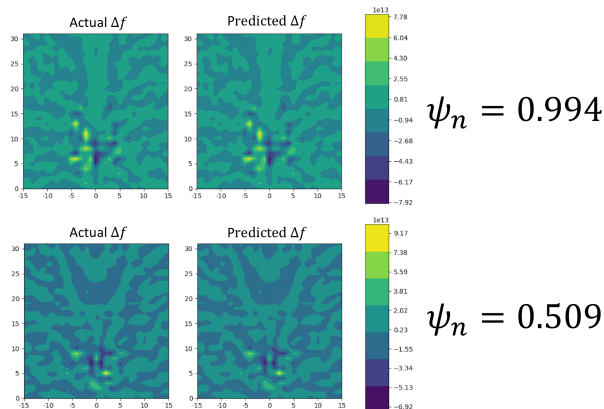
Figure 6: Comparison of Δf_{col} prediction in two different regimes of collisionality
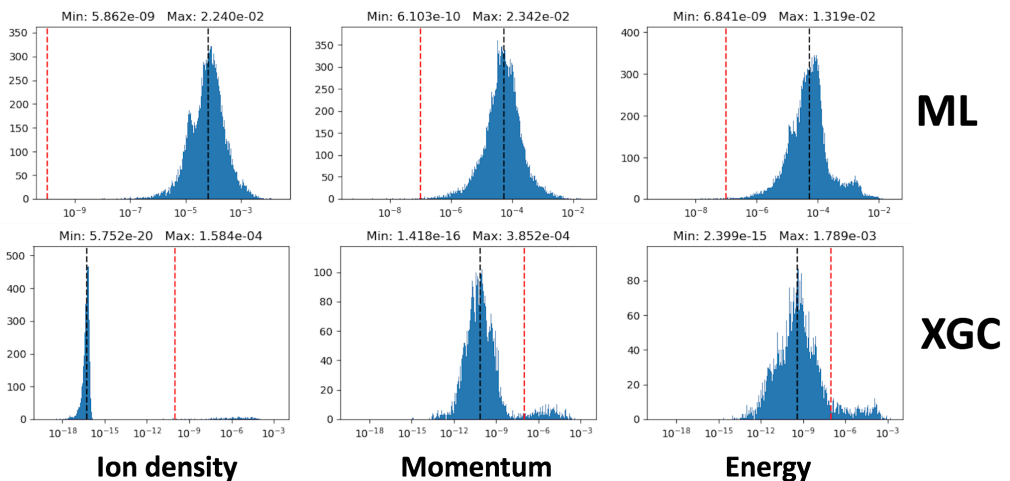

Figure 7: Comparison of conservation properties from XGC and from test set of neural network using machine learning (ML). The black vertical dashed lines represent the median value for the specific quantity. The red ones represent the target value (see Equation 4.7). Note: the x-axes (and y-axes) are different in both sets of plots

7. Conclusions and Further Work

This work presents an encoder-decoder neural network that shows great potential in replacing the nonlinear FPL collision operator. Through a penalization method, the network enforces the relative conservation properties to 10^{-4} level, while minimizing ℓ_2 error to 10^{-3} level (mimization of the conservation error is more important than the ℓ_2 error minimization due to the random nature of the collision operation). Enforcement of the conservation constraints in the training have shown to greatly improve performance, and progress towards the goal of the low numerical error needed in a gyrokinetic code like XGC as demonstrated.

Using the current trained model would result in a significant speedup of the collision kernel in the XGC code, especially with many plasma species. However, some improvement is needed to bring down the relative conservation accuracy to the 10^{-5} level for a

safe application, with the time-integrated error limited to several percent, over thousands of collision timesteps in case the error propagation is of drift type. The network has also proven successful in predicting this transformation for a wide range of training data that spans different regimes of collisionality. In short, this work has proven that machine learning can in fact serve as a useful aid for especially intricate calculations in numerical work.

One immediate observation is that it would be useful to explore a wider variety of neural network architectures. It could be the case that an architecture that has perhaps not been as effective for image recognition problems could perform very well for the prediction of colliding plasma species. Furthermore, the optimization routine used here to impose the "soft" constraints is fairly rudimentary and can certainly be refined to produce an even better trained network. Work to increase the sophistication of the optimization scheme in the form of an augmented Lagrangian method is underway and has shown promise in increasing the accuracy by two orders of magnitude. This work will be reported as a subsequent publication.

On the physics side, it will be important in the future to investigate an even broader collisionality regime that might be anticipated in a detached divertor plasma. Regardless, training has already been done on plasma both in the core and in the edge, which already spans a wide range of collisionality. This can be seen in Figure 6, which compares the prediction for points at two different places in the tokamak. It is also important to investigate the performance of the neural network for many species. Especially as the network begins to train on data from other plasma species, would new and separate training need to be done for different species? In the interest of reducing run-time, it is still necessary to make comparisons of run-time of using this neural network for inference in actual XGC production runs. All of these questions need to be addressed before the network can be expected to predict the Δf reliably enough for use in the multispecies XGC collision kernel.

Acknowledgements

REFERENCES

2016 CVC10: Semantic Segmentation Dataset.

ALOM, ZAHANGIR, TAHA, TAREK M, YAKOPCIC, CHRIS, WESTBERG, STEFAN, NASRIN, SHAMIMA & ASARI, VIJAYAN K 2017 Comprehensive Survey on Deep Learning Approaches .

12

Arvind T. Mohan, Lubbers, Nicholas, Livescu, Daniel & Chertkov, Michael 2017 EMBEDDING HARD PHYSICAL CONSTRAINTS IN CONVOLUTIONAL NEURAL NETWORKS FOR 3D TURBULENCE Arvind **0** (2015), 1–17.

Berg, Jens & Nyström, Kaj 2018 A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing* **317**, 28–41, arXiv: 1711.06464.

Beucler, Tom, Pritchard, Michael, Rasp, Stephan, Ott, Jordan, Baldi, Pierre & Gentine, Pierre 2019 Enforcing Analytic Constraints in Neural-Networks Emulating Physical Systems (2), arXiv: 1909.00912.

Dominski, J., Chang, C. S., Hager, R., Helander, P., Ku, S. & Yoon, E. S. 2019 Study of up–down poloidal density asymmetry of high- impurities with the new impurity version of XGCa. *Journal of Plasma Physics* **85** (5).

Hager, Robert, Yoon, E. S., Ku, S., D'Azevedo, E. F., Worley, P. H. & Chang, C. S. 2016 A fully non-linear multi-species Fokker-Planck-Landau collision operator for simulation of fusion plasma. *Journal of Computational Physics* **315**, 644–660.

He, Siyu, Li, Yin, Feng, Yu, Ho, Shirley, Ravanbakhsh, Siamak, Chen, Wei & Póczos, Barnabás 2019 Learning to predict the cosmological structure formation. *Proceedings of the National Academy of Sciences of the United States of America* **116** (28), 13825–13832, arXiv: 1811.06533.

Helander, Per & Sigman, Dieter J. 2002 *Collisional Transport in Magnetized Plasmas*. Cambridge University Press.

Jiang, Chiyu "Max", Prabhat, Kashinath, Karthik & Marcus, Phillip 2020 Enforcing Hard Physical Constraints in CNNs Through Differentiable PDE Layer .

Kao, J.C. 2017 Convolutional neural networks. Lecture notes at UCLA.

Karn, Ujjwal 2016 An Intuitive Explanation of Convolutional Neural Networks.

Ku, S., Chang, C. S., Hager, R., Churchill, R. M., Tynan, G. R., Cziegler, I., Greenwald, M., Hughes, J., Parker, S. E., Adams, M. F., D'Azevedo, E. & Worley, P. 2018 A fast low-to-high confinement mode bifurcation dynamics in the boundary-plasma gyrokinetic code XGC1. *Phys. Plasmas* **25** (5), 056107.

Landau, LD 1936 The transport equation in the case of coulomb interactions. *Collected papers of LD Landau* pp. 163–170.

Liu, Liyuan, Jiang, Haoming, He, Pengcheng, Chen, Weizhu, Liu, Xiaodong, Gao, Jianfeng & Han, Jiawei 2019 On the Variance of the Adaptive Learning Rate and Beyond pp. 1–14, arXiv: 1908.03265.

Qin, Hong 2019 Machine learning and serving of discrete field theories – when artificial intelligence meets the discrete universe pp. 1–25, arXiv: 1910.10147.

Ronneberger, Olaf, Fischer, Philipp & Brox, Thomas 2015 U-net: Convolutional networks for biomedical image segmentation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **9351**, 234–241, arXiv: 1505.04597.

Sanchez-Gonzalez, Alvaro, Bapst, Victor, Cranmer, Kyle & Battaglia, Peter 2019 Hamiltonian Graph Networks with ODE Integrators , arXiv: 1909.12790.

Simonyan, Karen & Zisserman, Andrew 2015 Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* pp. 1–14, arXiv: arXiv:1409.1556v6.

Sirignano, Justin & Spiliopoulos, Konstantinos 2018 DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics* **375** (Dms 1550918), 1339–1364, arXiv: 1708.07469.

Visin, Francesco, Romero, Adriana, Cho, Kyunghyun, Matteucci, Matteo, Ciccone, Marco, Kastner, Kyle, Bengio, Yoshua & Courville, Aaron 2016 ReSeg: A Recurrent Neural Network-Based Model for Semantic Segmentation. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* pp. 426–433, arXiv: 1511.07053.

Yoon, E. S. & Chang, C. S. 2014 A Fokker-Planck-Landau collision equation solver on two-dimensional velocity grid and its application to particle-in-cell simulation. *Physics of Plasmas* **21** (3).