Using Deep Neural Networks to Approximate Real Time Current Correlation Functions

Sophia Kivelson
Department of Computer Science
Stanford University
`skivelso@stanford.edu`

Abstract

In this paper I explore the use of deep convolutional networks and self attention to learn a mapping from simulated Quantum Monte Carlo ”imaginary time” data to physically measurable “real time” functions. In particular, I look at the current correlation problem, meaning the data used is designed to approximate an experiment investigating the behavior of current in a material.

1 Introduction

For many purposes, physicists want to be able to interpolate between interesting, physically measurable, dynamic properties of materials and theoretical descriptions of what is going in those material at a thermodynamic level. However, this interpolation (referred to as analytic continuation) is not trivial, and thus constitutes a barrier in a whole class of problems which relate thermodynamic properties to dynamical properties. For condensed matter theory in particular, making progress on any member of this class of problems would be a significant accomplishment.

In CS 221 I took a first stab at solving a particular member of this class of problems in which the dynamic property in question was the behavior of current in a material. To test hypotheses about the underlying thermodynamic theory behind the behavior of electrical currents, physicists create and study models of systems of interacting electrons and run Quantum Monte Carlo calculations (QMCC) on those models. The output of those calculations is the function, Λ , expressed as a series of real numbers corresponding to its values at discrete “imaginary times.” In principal there exists a one to one mapping between any Λ output and a conductivity function σ which is a function of a real frequency, ω; σ determines the current in a material in response to an applied electric field. Going from a calculated Λ to an experimetnally measurable σ is a problem of analytic continuation, which turns out to be intrinsically hard. The goal of the project was to learn a mapping from characteristic QMCC outputs, Λ ’s, to their associated σ’s.(9)

For this project, I revisit that current correlation problem, while changing the model used, the nature of the input data, and, consequently, the problem framing. Based on the hypothesis that any σ function can be well approximated as a linear combination of potentially simpler functions, I propose trying to learn a set of parameters which would specify three such ’simple’ functions which, when linearly combined, yield the true target σ.(4)(2)

2 Related work

The traditional technique to solve this kind of analytic continuation problem is the Max Entropy (MaxEnt) algorithm. Like most of the other approaches to this class of problem, given an imaginary time function like $\Lambda(\tau)$, MaxEnt uses regression to predict a dynamic function, like σ, directly.(4)(7)This kind of method can makes very reliable predictions but only for fairly smooth functions. It breaks down when σ has sharp features, which in many interesting cases, it does. (More on this in section 3.) Though my approach to the problem (i.e guessing σ component parameters and weights) differs significantly from the regression task MaxEnt seeks to solve, it does establish an important baseline for me. It indicates that, to count as progress on the problem, my model needs to be able to make accurate predictions when sharp features are present in $\sigma(\omega)$. (The discussion of "accurate predictions of sharp features" is picked up in section 5.)

Recently Song et al have implemented a tool for computing self energies, another thermodynamically determined dynamic property like σ, which relies on autoencoders.(1) Both their data generation process and training loop involve calculations made with something called Dynamic Mean Field Theory (DFMT) which many would argue introduces a new set of problems.(3) Moreover, the authors' choice to use DMFT reflects one of their overarching goals: trying to remove notions of domain bias from their solution. Indeed, this goal is shared among many who have tried to apply deep learning to analytic continuation problems. However, for my purposes, rather than trying to remove the constraints of paradigmatic physical framings, I intend to explicitly leverage domain knowledge to build a tool that is designed to aid physicists in their analysis. Furthermore, Song et al. employ ML (in the form of their auto-encoder) in just one step of a 5 step computation loop. These facts, and the fact that determining self energy tends to be a considerably easier problem, again distinguish our methods and problem farmings significantly.

Looking at the deep learning literature more broadly, I found that 1D convolutions had been used successful to learn information from time series functions, like in the case of echocardiograms.(6) Cordonnier et al's recent paper *On the Relationship between Self-Attention and Convolutional Layers* presents the argument that multi-head attention models can learn representations that are at least as expressive, if not more expressive, than those learned by convolutional neural networks (CNNs). (8) Based on this I chose to experiment with CNN's and attention heads in my model architecture.

3 Data

3.1 Underlying Physics and Understanding the Data

The goal of my project is to build a network able to map the output of QMCC to a real valued function of the sort that can be measured experimentally in a lab. The imagery time function Λ is characteristic of the kind of data a QMCC system would output.

In some archetypal cases, the behavior of a material is dominated by a single, theoretically well characterized process, for which a functional description of the measured σ and the corresponding Λ are known. More complicated (and interesting) materials display properties associated with more than one of these archetypal processes. Thus, more generally, it is reasonable to expect (and experimentally observed) that σ can often be well approximate as a linear combination of a few archetypal contributions to σ.

To begin with, I will consider a case in which σ is the sum of three functions, all with the same functional form but with different values of the parameters that control their behavior. The conductivity's of many metals can be well described this way.

In particular, for this project, I will experiment with a dataset where all the candidate σ's have the "Drude functional form," characteristic of a Fermi Liquid. The parameters in a Drude conductivity are an overall weight α and a parameter γ, known as the scattering rate, which determines the width in frequency for which the conductivity is significant. (For a continued discussion of my choice of material, see Appendix section 1).

3.2 Data Generation

A single Drude conductivity is described by the function

$$\sigma(\omega) = \frac{\gamma}{\omega^2 + \gamma^2} \; . \tag{1}$$

So as to have a model of the temperature dependence of γ, we split the generation of this parameter up into two terms. Together we represent a single γ parameter evaluated at temperature T as follows,

$$\gamma = \gamma_0 + \gamma_1 * T^2 \tag{2}$$

Each Λ in my dataset is a linear combination of the Λ's associated with three unique σ's of this form. We consider the problem in a range of temperatures, $T \in [0.1, 0.2, ..., 1.0]$ and we borrow the parameters determining σ to derive Λ at each T so that for each example:

$$\Lambda_T = \alpha_1 \Lambda_1(\gamma_{0_1}, \gamma_{1_1}, T) + \alpha_2 \Lambda_2(\gamma_{0_2}, \gamma_{1_2}, T) + \alpha_3 \Lambda_3(\gamma_{0_3}, \gamma_{1_3}, T) \tag{3}$$

where the coefficients α_i are sampled from range [0,1], and we ensure that $\sum_{i=1}^{3} \alpha_i = 1$. When generating data, we can choose 1 of 4 distributions from which to sample γ_0's and γ_1's. (Note that regardless of distribution, all γ's will be floats in the range 0 to 10.) The distribution options represent σ's which we expect to be hard to guess at high temperature, hard to guess at low temperatures, hard to guess at all temperatures or easy to guess at all temperatures. (For more on the distributions from which γ's are sampled, see section 3.3.)

Together, three sets of the parameters α_i , γ_{0_i} and γ_{1_i} comprise one label (as in equation (7)). Using the label, we can generate a training example. For each term $\Lambda_i(\tau)$ we evaluate the following equation at 200 values of τ (which range between 0 and 1/T).

$$\Lambda(\tau) = \frac{1}{\gamma} + F(\tau, 0) + F(\tau, \gamma) \tag{4}$$

$$F(\tau, 0) = -(\frac{1}{2\pi})(log(4\sin^2(\pi\tau)) + \epsilon \tag{5}$$

$$F(\tau, \gamma) = \sum_{i}^{\infty} \frac{\cos(2\pi i \tau)}{\gamma i + 2\pi i^2 + \epsilon} \tag{6}$$

where ϵ = 1/4000 and is included for numerical stability. I approximate (6) by summing the first 1000 terms. This 1000 comes from the parameter lambda_precision, This value can be changed to effectively increase or decrease data noise in future experiments.

We then take the element wise sum over all three 200-dimensional vectors produced this way, so that $\Lambda = \sum_{i=1}^{3} \alpha_i \Lambda_i$. We do this at 10 temperatures so that each input is a 10 by 200 matrix of floats. We can represent Λ evaluated at each T as Λ_T. So finally, each training example ends up as a 10 by 200 matrix, with a 3 by 3 dimensional label formatted as follows

$$\begin{bmatrix} \Lambda_{0.1}(\tau_1) & \Lambda_{0.1}(\tau_2) & ... & \Lambda_{0.1}(\tau_{200}) \\ \Lambda_{0.2}(\tau_1) & \Lambda_{0.2}(\tau_2) & ... & \Lambda_{0.2}(\tau_{200}) \\ ..... & & & \\ \Lambda_{1.0}(\tau_1) & \Lambda_{1.0}(\tau_2) & ... & \Lambda_{1.0}(\tau_{200}) \end{bmatrix} \begin{bmatrix} \alpha_1 & \gamma_{0_1} & \gamma_{1_1} \\ \alpha_2 & \gamma_{0_2} & \gamma_{1_2} \\ \alpha_3 & \gamma_{0_3} & \gamma_{1_3} \end{bmatrix} \tag{7}$$

3.3 Datasets

I have three datasets on which I ran experiments. Each dataset contains 2,000 training examples, 200 validation examples and 200 test examples. What varies across the datasets are the constraints placed on the sampling of γ_0 and γ_1. The particulars of each set of constraints are elaborated on Figure 1 of the Appendix.

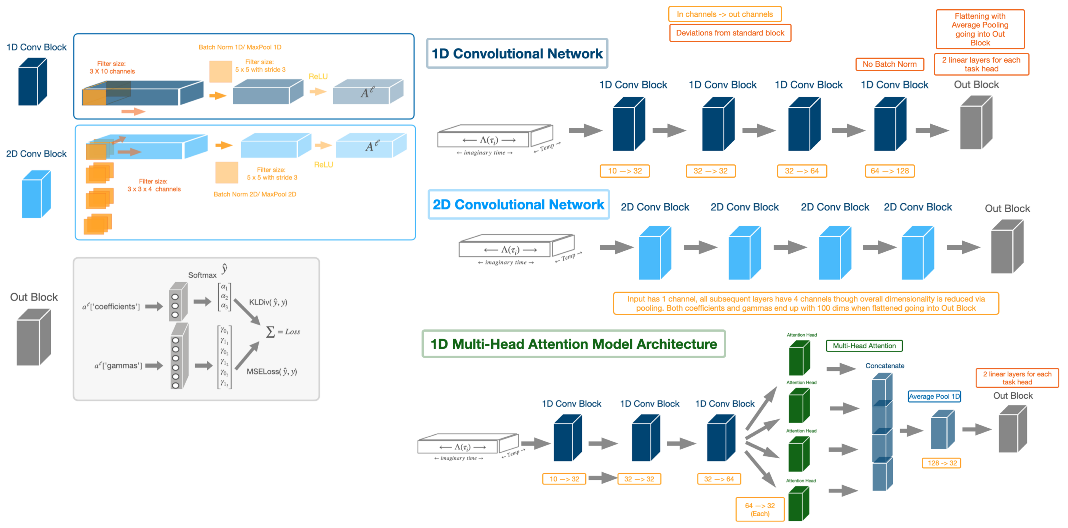
The first data set is "easy_data" in which, for each example, $\sigma_1, \sigma_2, \sigma_3$ are all chosen to have γ's which satisfy the constraints listed under 'Easy at all T' in Figure 1. The second is "mixed_data" in which, for each example $\sigma_1, \sigma_2, \sigma_3$ are all sampled from the same distribution. In this data set that common distribution was randomly chosen for each example from the set [Easy at all T, Hard at high T, Hard at Low T]. The final dataset is "element_mixed_data" in which, for each example, σ_1 is sampled from the 'Easy at all T' distribution, σ_2 is sampled from the 'Hard at high T' distribution and σ_3 is sampled from 'Hard at low T' distribution.

4 Models

The different manner in which the label elements were generated suggested a multitask approach. The α_i's are evaluated at one task head, while the γ_{0_i} and γ_{1_i}'s are evaluated at the second task head. Unless otherwise specified, in each architecture I experimented with, each task head consists of one additional Fully Connected Layer as depicted in the Out Block details in the image bellow.(6)

Leading up to the Out Block, after experimenting a little with various depths and hyper parameters, I chose the architectures depicted bellow to respectively use as the 1D Convolutional Network, the

2D Convolutional Network and the Attention Network in my experiments comparing model types. I did not spend much time tuning hyper parameters as, for the scope of this project, I was primarily interested in starting a high level investigation into the capabilities of different models constrained to similar sized embedding spaces. (Note that, in the interest of space, the details of the Attention Head can be found in figure 2 in the appendix, as can a blown up version of this image for better readability)


1D Conv Block
Batch Norm 1D/ MaxPool 1D
Filter size: 3 X 10 channels
Filter size: 5 x 5 with stride 3
ReLU
A^{ℓ}
2D Conv Block
Filter size: 3 x 3 x 4 channels
Filter size: 5 x 5 with stride 3
ReLU
Batch Norm 2D/ MaxPool 2D
A^{ℓ}
Out Block
Softmax
$\hat{y}$
a^{ℓ}['coefficients']
a^{ℓ}['gammas']
KLDiv($\hat{y}$, y)
MSELoss($\hat{y}$, y)
$\sum = Loss$
In channels -> out channels
Deviations from standard block
Flattening with Average Pooling going into Out Block
No Batch Norm
2 linear layers for each task head
1D Convolutional Network
$\Lambda(\tau_i)$
imaginary time
Temp
1D Conv Block
Out Block
10 —> 32
32 —> 32
32 —> 64
64 —> 128
2D Convolutional Network
2D Conv Block
Input has 1 channel, all subsequent layers have 4 channels though overall dimensionality is reduced via pooling. Both coefficients and gammas end up with 100 dims when flattened going into Out Block
1D Multi-Head Attention Model Architecture
Attention Head
Multi-Head Attention
Concatenate
Average Pool 1D
128 -> 32
64 —> 32 (Each)

4.1 Loss

The α_i's can be thought of as representing a probability distribution over the three constituent functions. Thus I use KL-Divergence as the loss for this task head. As the α_i's sum to one 1 I apply a softmax before calculating the Loss. As each γ_{0_i} and γ_{1_i} are real numbers between 0 and 10, I simply use mean squared error for the second task head.

If I were to plot $\sigma(\omega)$ for a given training example there are two aspects a physicist would care about getting right: the shape of the curve and its behavior at $\omega = 0$. Close to $\sigma(0)$ is where we see the sharp function features mentioned in the related work section that are particularly hard to get right. Thus, I experimented with adding a term to the loss which explicitly incentivizes precision at $\sigma(0)$, calculated as follows:

$$\sigma(0) = \frac{\alpha_1}{\gamma_1} + \frac{\alpha_2}{\gamma_2} + \frac{\alpha_3}{\gamma_3} \tag{8}$$

Where γ is computed according to equation (2). Using this I define three different loss functions. In the following assume that hat indicates a predicted value, $\vec{\alpha}$ is a 3 dimensional vector containing all three α_i's, $\vec{\gamma}$ is a 6 dimensional vector containing all three pairs of $\gamma_{0_i}, \gamma_{1_i}$ stacked linearly.

Loss 1: $KLDiv(\vec{\hat{\alpha}}, \vec{\alpha}) + MSE(\vec{\hat{\gamma}}, \vec{\gamma})$

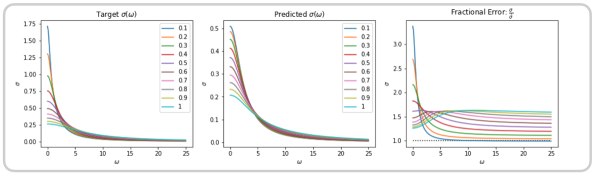
Loss 2: $0.9(\textbf{Loss1}) + 0.1(MSE(\sigma(\hat{0}), \sigma(0)))$

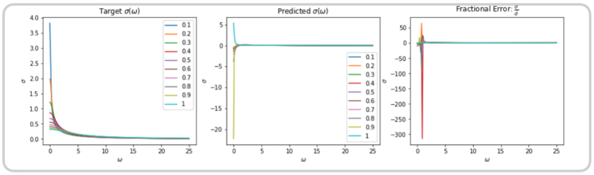
Loss 3: $0.5(\textbf{Loss1}) + 0.5(MSE(\sigma(\hat{0}), \sigma(0)))$

5 Experiments and Conclusions

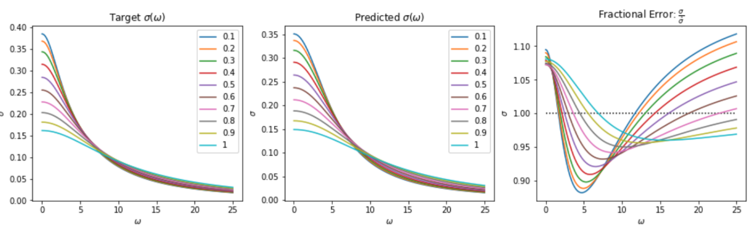
There are three elements which vary across experiments - 1. the model used, 2. the data used and 3. the loss function used. Furthermore, there are two ways in which I determine the success or failure each experiment. The first is the usual - by looking at losses and inspecting predictions. The second is by looking at the shape and precision per frequency of $\sigma(\omega)$. To do so, I graph something called the fractional error of $\sigma(\omega)$. The fractional error is represented as a plot of $\frac{\sigma(\omega)}{\hat{\sigma}(\omega)}$ vs ω. In the ideal case, for all temperatures, the plot of fractional error would contain only straight lines at $\frac{\sigma(\omega)}{\hat{\sigma}(\omega)} = 1$. Still, if a model could consistently get $\frac{\sigma(\omega)}{\hat{\sigma}(\omega)}$ in the range of about 0.8 to 1.2 for all temperatures, it would be a significant accomplishment and usable in real physics experiments.

For reference here is an example of 2 sets of graphs with which we can compare predicted vs target σ's. They are organized into sets of three where from left to right you see the plot of the true target $\sigma(\omega)$, the predicted $\sigma(\omega)$ and the fractional error, all plotted against frequency, ω. To contextualize the discussion, the left cell shows graphs associated with an average to good result where as the right shows graphs associated with a very bad prediction.





Training on the easy_data dataset, all of the models were able to guess γ_{0_i}'s and γ_{1_i}'s reasonably well (i.e. made guesses that where within ± 1 of the target) for many, if not most examples. However, the models where not able to get much of a signal on α_i's, guessing close to a 0.33 weighting for each component σ across all examples. A similar if slightly less pronounced effect happened when training on mixed_data. The attention model was slightly unique in this as, for all datasets, but particularly when trained on the element_mixed_data, it had a tendency to put all the weight in one component σ such that one of $\sigma_1, \sigma_2, \sigma_3$ ended up being very close to 1 while the other two where close to zero. These factors suggests that when all the γ parameters are sampled from the same distribution, it does not result in enough diversity to, when linearly combined, produce a new σ which is outside of that distribution. My approach in this case is probably not as useful as in cases in which the true sigma is a more complex function. It also suggests that it may be worthwhile to try and use the attention model to guess the true σ functions directly.



It is also worth noting that, despite not getting the parameter values correct, on the easy_data especially, all of the models where able to do fairly well in guessing the form of the σ function as illustrated here.

Overall I would say that the Attention Network performed best. It consistently predicted at least a third of the examples in the hardest dataset, element_mixed_data, correctly within a use-able range (i.e. the fractional error more or less stayed in the range 0.8 to 1.2), and was able to do so regardless of the loss function used.

The 2D Convolutional Network when trained with Loss 1, performed as well as the Attention Network, if not better on certain runs. This includes equal performance in terms of predicting near zero frequency values of σ. However, interestingly the model which performed by far the worst was also the 2D Convolutional Network when it was trained with loss 3. It will take a little more investigation to figure out what this means, but I think it suggests something very philosophically interesting about the nature of the information contained in different sections of the data. The 1D Convoltional Network was much less sensitive to changes in the loss function, re-enforcing the idea that the 2D spacial decomposition is extracting potentially quite different information from the other two models. (A visual comparison of the experimental results of the 2D Conv Network and the Attention Network can be found in Appendix Figure 3.)

I am overall very pleased with the results I have gotten thus far an intend to continue to experiment with Attention, 2D convolutions and tweaks to the data and problem framing.

References

[1] Song, Taegeun, et al. “Analytic Continuation of the Self-Energy via Machine Learning Techniques.” ArXiv.org, 27 July 2020, arxiv.org/abs/2007.13610.

[2] R.N. Silver, D. S. Sivia, and J. E. Guber- natis, “Maximum-entropy method for analytic continuation of quantum Monte Carlo data,” Phys. Rev. B 41, 2380 (1990)

[3] Vollhardt, Dieter, et al. “Dynamical Mean-Field Theory.” Cond-Mat.str-El], 22 Sept. 2011.

[4] Mehryar. Lecture: Foundations of Machine Learning Maximum En- tropy Models, Logistic Regression. Https://Cs.nyu.edu/ Mohri/Mls/ml maxent models.Pdf, NYU and Courant Institute and Google Re- search.

[5] Yoon, Hongkee Sim, Jae-Hoon Han, Myung. (2018). Analytic continuation via ’domain-knowledge free’ machine learning.

[6] Khan, A., Sohail, A., Zahoora, U. et al. A survey of the recent architectures of deep convolutional neural networks. Artif Intell Rev 53, 5455–5516 (2020). https://doi.org/10.1007/s10462-020-09825-6

[7] Jarrell, Maximun Entropy and Analytic Continuation. Lectures on the Physics of Strongly Correlated Systems XII, www.phys.lsu.edu/ jarrell/Green/MEM_Salerno.pdf.

[8] Cordonnier, Jean-Baptiste, et al. “On the Relationship between Self-Attention and Convolutional Layers.” ICLR 2020 Conference, 25 Sept. 2019.

[9] Kivelson, Sophia. Using NN’s to Approximate Real Time Current Correlation Functions. CS 221 Final Project.

[10] Venkatachalam, Mahendran. “Attention in Neural Networks.” Medium, Towards Data Science, 7 July 2019, towardsdatascience.com/attention-in-neural-networks-e66920838742.

Appendix

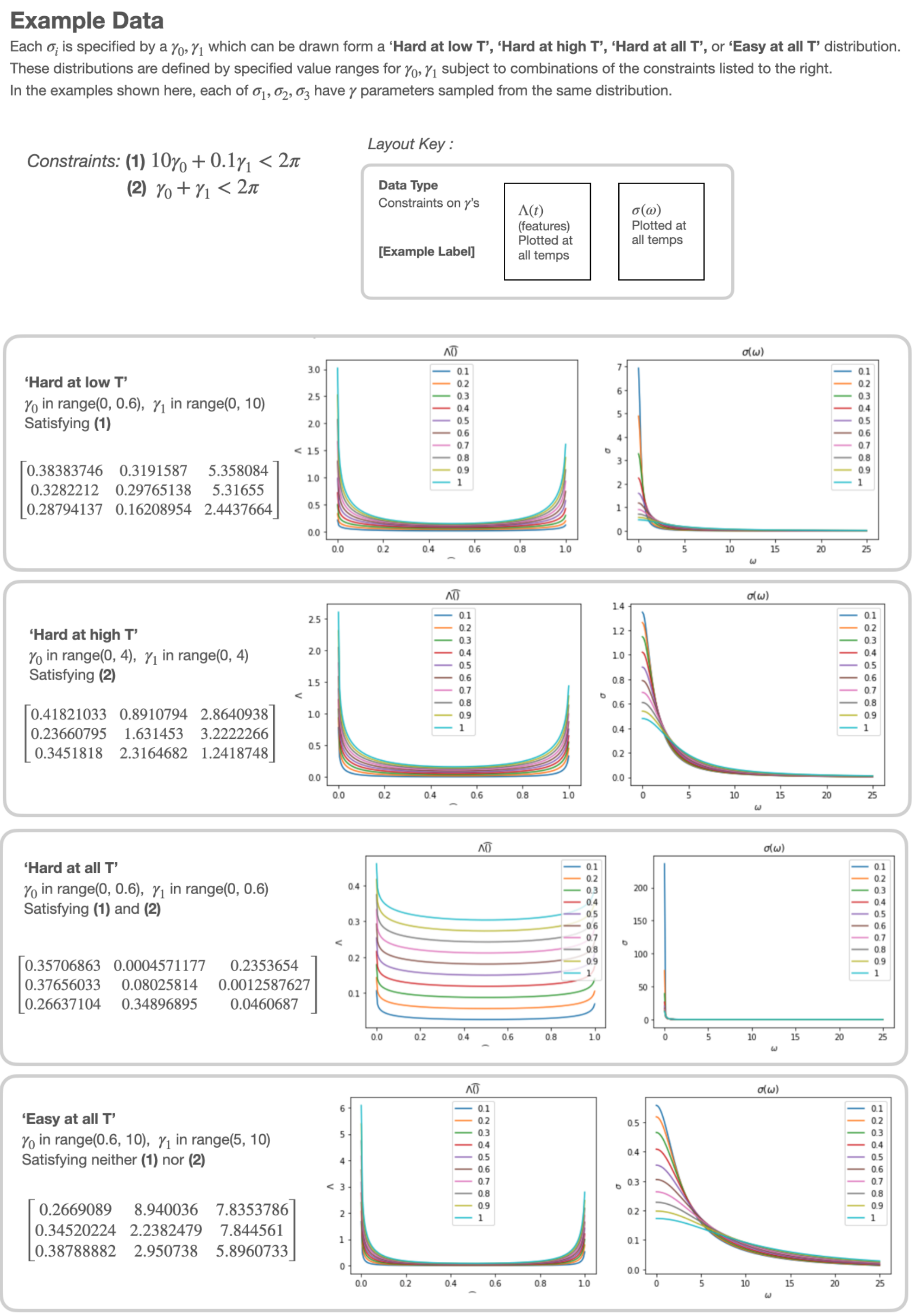
A Why Fermi Liquids?

Fermi Liquids are desirable for my purposes for a few reasons. Firstly, by varying the parameter γ, we can find $\sigma(\gamma)$’s which exhibit all the behaviors that make the Λ to σ mapping challenging. The first major one of these challenges is the presence of sharp features in σ, where by sharp features I mean having regions in which the variation in σ is large compared to the width of the interval over which ω is varied (in units of the temperature T). This occurs whenever γ is small compared to the temperature. The second is that if I were to define $\sigma = \sum_{i=1} \alpha_i \sigma(\gamma_i)$ - i.e. if I had a some Fermi Liquid in which the behavior of current could be given as a linear combination of the σ’s of differently parameterized Fermi Liquids (weighted by coefficients α) - then I could expect σ to have the following properties: Firstly, if any of the σ_i’s in the sum are ’hard’ (have small γ’s), then the resulting σ will also be a ’hard’ example to classify. Secondly, even if the α_i weighting that ’hard’ σ_i were small, it is possible for it to have a very disproportionate effect on σ. In addition to capturing the characteristic challenges of this current correlation problem, for any σ of this form, we can solve for Λ given a σ with a relatively simple equation. This means, using linear combinations of differently parameterized Fermi Liquid σ functions, I can relatively easily create a training dataset mapping between Λ’s and σ’s which captures the intricacies of the true problem. In other words, if I can build a model that performs well on this kind of dataset, then that model is very likely to be usable on generic QMCC data.

B Figures

Figure 1

Example data with annotations for how to sample γ_0 amd γ_1



Example Data

Each σ_i is specified by a γ_0, γ_1 which can be drawn form a '**Hard at low T**', '**Hard at high T**', '**Hard at all T**', or '**Easy at all T**' distribution. These distributions are defined by specified value ranges for γ_0, γ_1 subject to combinations of the constraints listed to the right. In the examples shown here, each of $\sigma_1, \sigma_2, \sigma_3$ have γ parameters sampled from the same distribution.

Constraints: **(1)** $10\gamma_0 + 0.1\gamma_1 < 2\pi$
(2) $\gamma_0 + \gamma_1 < 2\pi$

Layout Key :

Data Type
Constraints on γ's

[Example Label]

$\Lambda(t)$ (features) Plotted at all temps

$\sigma(\omega)$ Plotted at all temps

'Hard at low T'
γ_0 in range(0, 0.6), γ_1 in range(0, 10)
Satisfying **(1)**

$$\begin{bmatrix} 0.38383746 & 0.3191587 & 5.358084 \\ 0.3282212 & 0.29765138 & 5.31655 \\ 0.28794137 & 0.16208954 & 2.4437664 \end{bmatrix}$$

'Hard at high T'
γ_0 in range(0, 4), γ_1 in range(0, 4)
Satisfying **(2)**

$$\begin{bmatrix} 0.41821033 & 0.8910794 & 2.8640938 \\ 0.23660795 & 1.631453 & 3.2222266 \\ 0.3451818 & 2.3164682 & 1.2418748 \end{bmatrix}$$

'Hard at all T'
γ_0 in range(0, 0.6), γ_1 in range(0, 0.6)
Satisfying **(1)** and **(2)**

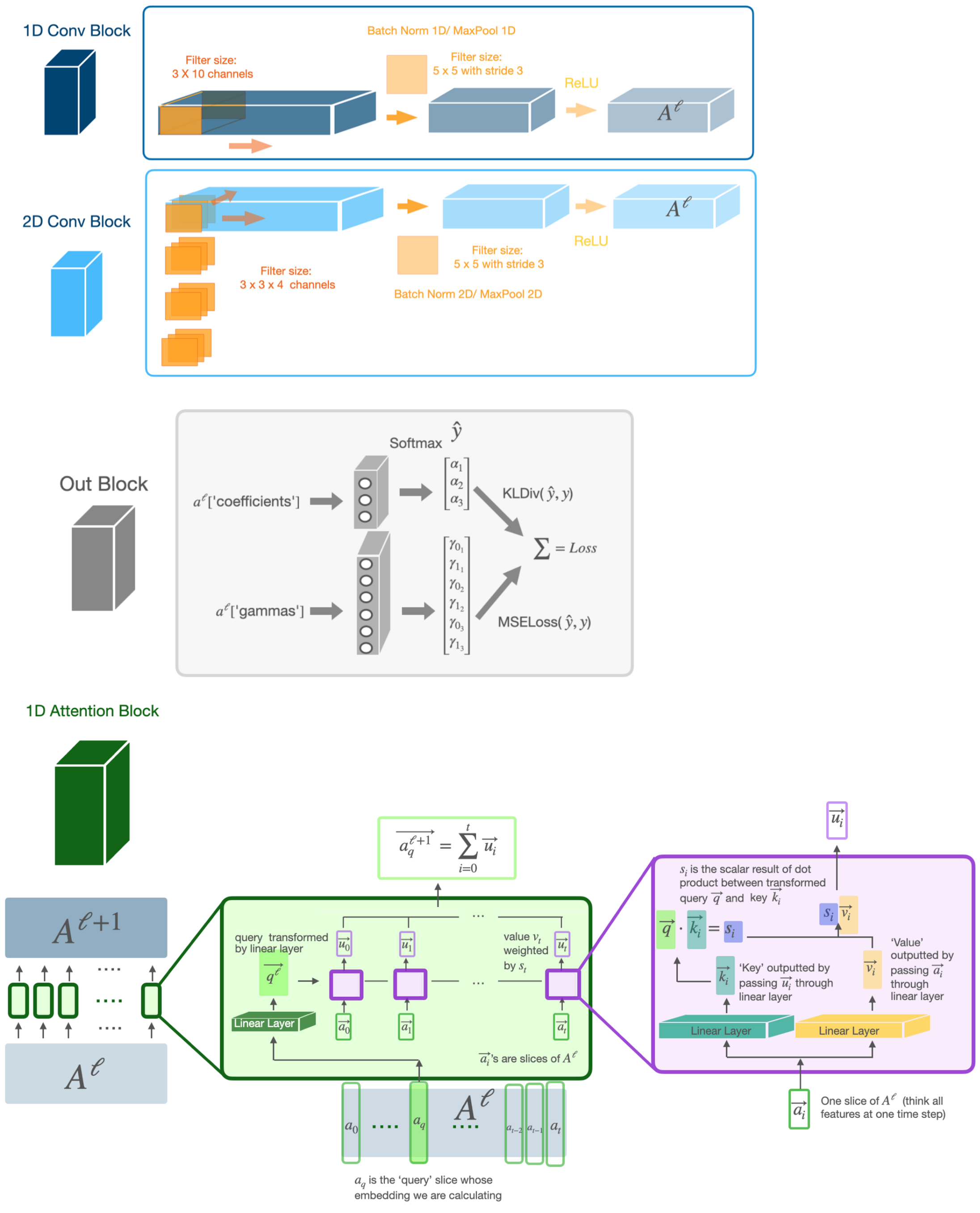
$$\begin{bmatrix} 0.35706863 & 0.0004571177 & 0.2353654 \\ 0.37656033 & 0.08025814 & 0.0012587627 \\ 0.26637104 & 0.34896895 & 0.0460687 \end{bmatrix}$$

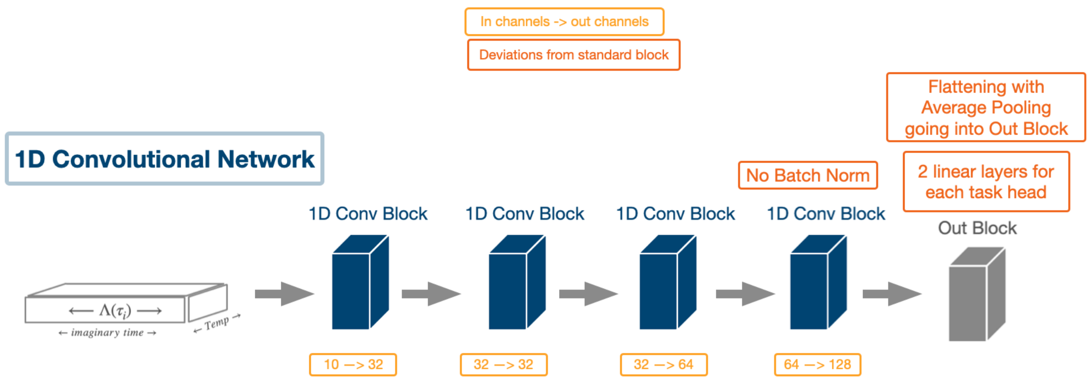
'Easy at all T'
γ_0 in range(0.6, 10), γ_1 in range(5, 10)
Satisfying neither **(1)** nor **(2)**

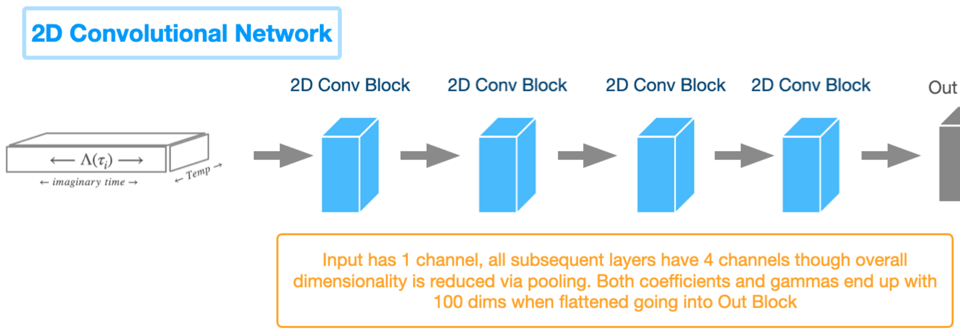
$$\begin{bmatrix} 0.2669089 & 8.940036 & 7.8353786 \\ 0.34520224 & 2.2382479 & 7.844561 \\ 0.38788882 & 2.950738 & 5.8960733 \end{bmatrix}$$

Figure 2

Model building blocks and architectures


1D Conv Block
Batch Norm 1D/ MaxPool 1D
Filter size: 3 X 10 channels
Filter size: 5 x 5 with stride 3
ReLU
A^ℓ
2D Conv Block
Filter size: 3 x 3 x 4 channels
Filter size: 5 x 5 with stride 3
Batch Norm 2D/ MaxPool 2D
ReLU
A^ℓ
Out Block
a^ℓ['coefficients']
a^ℓ['gammas']
Softmax
$\hat{y}$
α_1 α_2 α_3
γ_{0_1} γ_{1_1} γ_{0_2} γ_{1_2} γ_{0_3} γ_{1_3}
KLDiv($\hat{y}, y$)
MSELoss($\hat{y}, y$)
$\sum = Loss$
1D Attention Block
$A^{\ell+1}$
A^ℓ
$\overrightarrow{a_q^{\ell+1}} = \sum_{i=0}^{t} \overrightarrow{u_i}$
query transformed by linear layer
$\overrightarrow{q^\ell}$
Linear Layer
$\overrightarrow{u_0}$ $\overrightarrow{u_1}$ $\overrightarrow{u_t}$
$\overrightarrow{a_0}$ $\overrightarrow{a_1}$ $\overrightarrow{a_t}$
value v_t weighted by s_t
$\overrightarrow{a_i}$'s are slices of A^ℓ
a_0 a_q a_{t-2} a_{t-1} a_t
a_q is the 'query' slice whose embedding we are calculating
$\overrightarrow{u_i}$
s_i is the scalar result of dot product between transformed query $\overrightarrow{q}$ and key $\overrightarrow{k_i}$
$\overrightarrow{q} \cdot \overrightarrow{k_i} = s_i$
s_i $\overrightarrow{v_i}$
'Key' outputted by passing $\overrightarrow{u_i}$ through linear layer
'Value' outputted by passing $\overrightarrow{a_i}$ through linear layer
Linear Layer
Linear Layer
$\overrightarrow{a_i}$
One slice of A^ℓ (think all features at one time step)




2D Convolutional Network
2D Conv Block
2D Conv Block
2D Conv Block
2D Conv Block
Out Block
$\Lambda(\tau_i)$
imaginary time
Temp
Input has 1 channel, all subsequent layers have 4 channels though overall dimensionality is reduced via pooling. Both coefficients and gammas end up with 100 dims when flattened going into Out Block

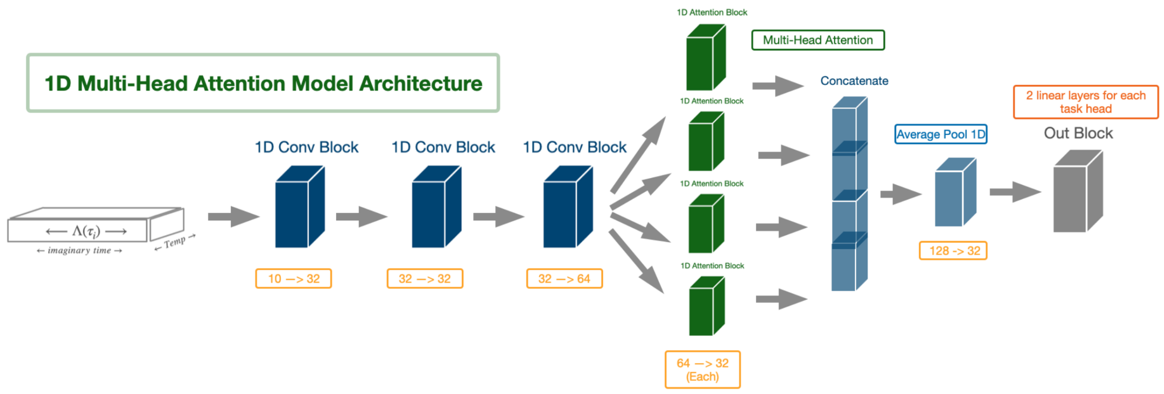
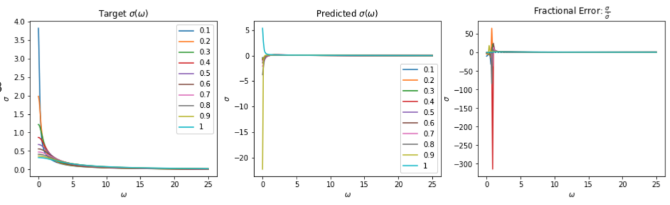


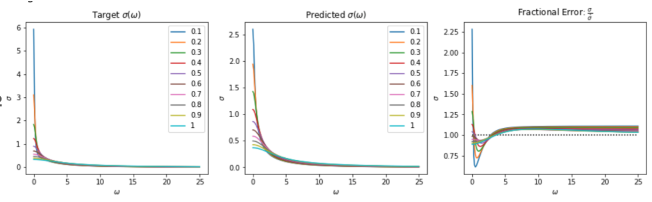
Figure 3
Experiment Results: Shown below are characteristic results from training each model with the loss listed.

2D Conv

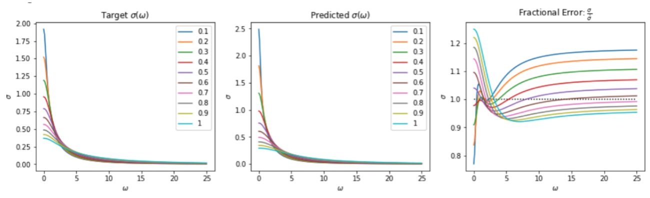
Loss 3



Loss 2

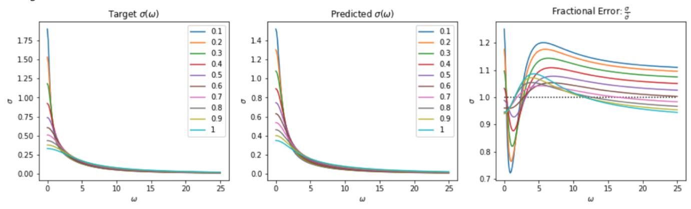


Loss 1

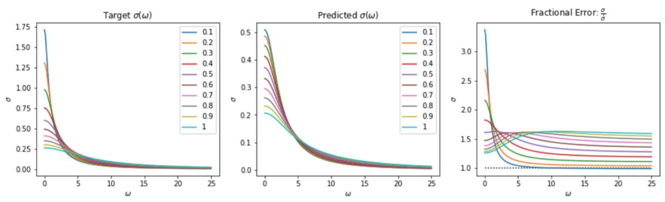


Attention

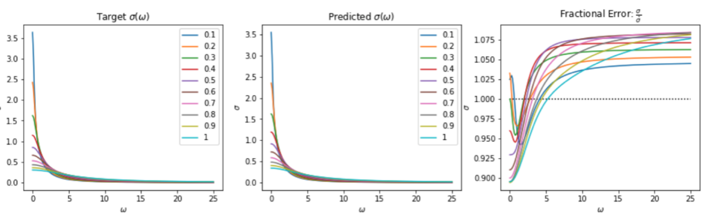
Loss 3



Loss 2



Loss 1


Target $\sigma(\omega)$
Predicted $\sigma(\omega)$
Fractional Error: $\frac{\hat{\sigma}}{\sigma}$
σ
ω