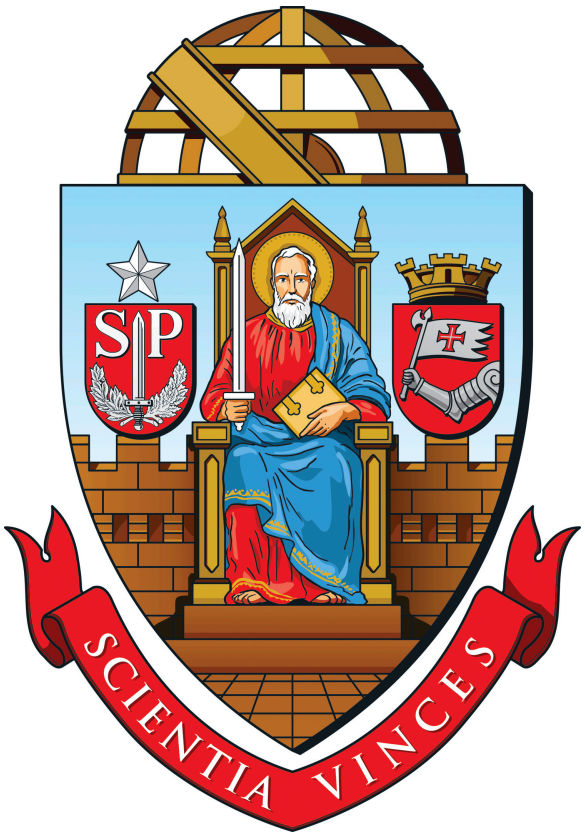
Relatório Parcial - PIBIC

Luan de Souza Silva 12557098 - IFUSP



Contents

1. Mecânica Quântica e Funções de Green

1.1 Revisão de Mecânica Quântica

Na Mecânica Quântica uma partícula é representada por estados (vetores ou "kets") no Espaço de Hilbert ($|\psi\rangle$). Os observáveis (energia, momento, etc) são representados por operadores hermitianos que atuam nesses estados. Daí obtemos um problema de autovalores e autovetores. Por exemplo:

$$\hat{H}\,|\psi\rangle = E\,|\psi\rangle$$

Note que como os observáveis representados por operadores hermitianos, os valores de E são sempre reais e os autovetores são ortogonais, de forma que podemos formar uma base no Espaço de Hilbert em que o operador $\hat{H}$ é diagonal. Para isso basta assegurar que os vetores de base $\{\phi_\alpha\}$ satisfaçam a relação de completeza:

$$\sum_\alpha |\phi_\alpha\rangle\,\langle\phi_\alpha| = I$$

Se já soubermos o estado em que se encontra o sistema, podemos calcular o valor esperado usando

$$\langle \hat{A} \rangle = \langle \psi | \, \hat{A} \, | \psi \rangle \tag{1}$$

Que no espaço de posição pode ser escrito como

$$\langle \hat{A} \rangle = \int \psi^* (\hat{A} \psi) \mathrm{d}\vec{r}$$

A equação (1) é válida apenas para o Estado Fundamental. Para temperaturas finitas temos

$$\langle \hat{A} \rangle = \frac{1}{Z} \operatorname{Tr}\Big\{ \hat{\rho} \hat{A} \Big\} \tag{2}$$

Em que Z é a função de partição do sistema e $\hat{\rho}$ é o operador densidade, que no Ensemble Canônico pode ser escrito como $\hat{\rho} = e^{\beta \hat{H}}$.

Para lidar com evolução temporal do sistema utilizamos a Equação de Schrodinger Dependente do Tempo:

$$i\hbar \frac{\partial \, | \psi \rangle}{\partial t} = \hat{H} \, | \psi \rangle$$

Note que no caso mais simples em que a Hamiltoniana não depende do tempo, a evolução temporal dos estados pode ser escrita como

$$| \psi(t) \rangle = e^{-i\hat{H}t/\hbar} \, | \psi(0) \rangle$$

Apesar de toda a descrição feita acima ser muito útil à primeira vista, ela não é tão simples de se usar quando o sistema envolve várias partículas. Digamos que um sistema seja composto de n partículas, em que as bases do Espaço de Hilbert de cada uma são $\{|\phi_\alpha\rangle\}_i$. Então a base do espaço de Hilbert de todo o sistema é dada por

$$\{|\Phi_\alpha\rangle\} = \bigotimes_{i=1}^{n} \{|\phi_\alpha\rangle\}_i$$

Uma forma mais operacional de se calcular esta base é usando o Determinante de Slater (para férmions).

No entanto, uma forma mais prática de se tratar deste tipo de sistema é usando uma base de Número de Ocupação e o Operador Número, bem como operadores de criação e aniquilação, descritos pela Segunda Quantização. Temos

$$\hat{n} = \hat{c}^\dagger \hat{c}$$

$$\hat{c} \, | 0 \rangle = 0$$

$$\hat{c}^\dagger \, | 0 \rangle = | 1 \rangle$$

Além disso, a álgebra desses operadores é descrita por várias relações que utilizam anticomutadores. Seguem abaixo as principais relações:

$$\{\hat{c}_i, \hat{c}_k\} = \{\hat{c}_i^\dagger, \hat{c}_k^\dagger\} = 0$$

$$\{\hat{c}_i, \hat{c}_k^\dagger\} = \delta ik$$

Neste formalismo podemos escrever a Hamiltoniana de um modelo Tight Binding para primeiros vizinhos como

$$\boxed{\hat{H} = \sum_i \epsilon_i \hat{n}_i - \sum_j t_{j+1,j} \hat{c}^\dagger_{j+1} \hat{c}_j + h.c} \tag{3}$$

Para o caso interagente, podemos usar o Modelo de Hubbard (que também se assemelha ao modelo de impureza de Anderson):

$$\boxed{\hat{H} = \sum_{i\sigma} \epsilon_i \hat{n}_{i\sigma} - \sum_{j\sigma} t_{j+1,j} \hat{c}^\dagger_{j+1\sigma} \hat{c}_{j\sigma} + h.c + U \sum_k \hat{n}_{k\uparrow} \hat{n}_{k\downarrow}} \tag{4}$$

Outro modelo de interesse é o Modelo de Hubbard-Holstein.

Estes últimos dois modelos mencionados são modelos interagentes, e serão explorados futuramente no projeto, com uso de DMRG.

Particularmente o Modelo de Impureza de Anderson é interessante devido ao Efeito Kondo, em que uma impureza magnética fica "'blindada" e seu momento magnético é suprimido.

1.2 Pictures de evolução temporal

Existem várias várias representações para o estudo da evolução temporal dos sistemas quânticos. A que foi apresentada anteriormente é a Representação de Schrodinger.

Na Representação de Schrodinger os operadores são independentes do tempo (exceto nos casos em que há dependência explícita) e os estados é que evoluem. Para o caso em que a hamiltoniana não depende do tempo, temos

$$|\psi(t)\rangle = e^{-i\hat{H}t/\hbar} |\psi(0)\rangle$$

Quando a Hamiltoniana é dependente do tempo, usamos a Série de Dyson.

No entanto, esta não é a única forma de se tratar este tipo de problema. Além da Representação de Schrodinger existem mais duas representações: a de Heiseinberg e a de Interação. Estas são elucidadas abaixo:

- Representação de Heisenberg:

  Nesta representação os estados são independentes do tempo e os operadores evoluem no tempo, de forma a obedecer a Equação de Heisenberg:

  $$i\hbar \frac{\mathrm{d}\hat{A}}{\mathrm{d}t} = [\hat{A}, \hat{H}] + i\frac{\partial \hat{A}}{\partial t}$$

- Representação de Interação:

  Seja um sistema perturbado tal que $\hat{H} = \hat{H}_0 + \hat{V}$, em que $\hat{V}$ é um potencial perturbativo e de interação. Assim, definimos

  $$|\psi_I(t)\rangle \equiv e^{i\hat{H}_0 t/\hbar} |\psi_S(t)\rangle$$
  $$\hat{A}_I(t) \equiv e^{i\hat{H}_0 t/\hbar} \hat{A}_S e^{-i\hat{H}_0 t/\hbar}$$

  De forma que as seguintes equações são satisfeitas:

  $$i\hbar \frac{\partial |\psi_I\rangle}{\partial t} = \hat{V}_I |\psi_I(t)\rangle$$
  $$\frac{\mathrm{d}\hat{A}_I}{\mathrm{d}t} = \frac{i}{\hbar}[\hat{H}_0, \hat{A}_I(t)]$$

1.3 Funções de Correlação e Funções de Green Quânticas

Uma função de Correlação dependente do tempo é escrita como

$$\langle \hat{A}(t)\hat{B}(t')\rangle = \frac{1}{Z}\,\mathrm{Tr}\Big\{\hat{\rho}\hat{A}(t)\hat{B}(t')\Big\}$$

No caso de funções de correlação retardadas, temos $t > t'$ e escrevemos

$$C^r_{A,B}(t-t') = -i\theta(t-t')\langle \hat{A}(t)\hat{B}(t')\rangle$$

Podemos ainda tratar este problema no domínio da frequência via Tranformada de Fourier, tomando $\omega \to \omega^+ = \omega + i\eta$, em que $\eta \to 0$, de forma que

$$C^r_{A,B}(\omega^+) = \int_{-\infty}^{\infty} C^r_{A,B}(t-t')e^{i\omega^+(t-t')}\,\mathrm{d}(t-t')$$

As Funções de Green são funções de correlação em que os operadores $\hat{A}$ e $\hat{B}$ são os operadores de campo (e.g operadores de criação e aniquilação):

$$\boxed{G^R(\vec{r},t,\vec{r}',t') = -i\theta(t-t')\langle[\hat{\psi}(\vec{r},t),\hat{\psi}^\dagger(\vec{r}',t')]_\pm\rangle}$$

Também é comum definir mais outras "formas" de funções de Green:

$$\text{Greater GF: } G^> = -i\langle\hat{\psi}\hat{\psi}^\dagger\rangle$$

$$\text{Lesser GF: } G^< = -i(\pm 1)\langle\hat{\psi}^\dagger\hat{\psi}\rangle$$

As funções de Green são de grande importância pois, através da Representação de Lehmann sabemos que elas estão relacionadas à função Densidade de Estados (Função Espectral):

$$A_i(\omega^+) = \pm\frac{1}{\pi}\,\mathrm{Im}\big\{G^R_{ii}(\omega^+)\big\} \tag{5}$$

Esta função é de particular interesse para o cálculo de propriedades eletrônicas dos sistemas, como número de ocupação, condutância, etc.

Para estudar a evolução temporal das GF usamos a técnica de Equação de Movimento(1):

$$i\frac{\partial}{\partial t}G^R_{km}(t-t') = \delta_{km}\delta(t-t') - i\theta(t-t')\langle[[\hat{a}_k(t),\hat{H}],\hat{a}^\dagger_m(t')]_\pm\rangle$$

Que para operadores fermiônicos se torna

$$i\frac{\partial}{\partial t}G^R_{km}(t-t') = \delta_{km}\delta(t-t') - i\theta(t-t')\langle\{[\hat{c}_k(t),\hat{H}],\hat{c}^\dagger_m(t')\}\rangle$$

Ou, na Notação de Zubarev

$$\boxed{i\frac{\partial}{\partial t}G^R_{km}(t-t') = \delta_{km}\delta(t-t') + \langle\langle[\hat{c}_k(t),\hat{H}] : \hat{c}^\dagger_m(t')\rangle\rangle_{(t-t')}} \tag{6}$$

Com isso podemos calcular a evolução temporal das funções de Green de um sistema, e então utilizar FT para passar para o domínio da frequência.

1.4 Tight Binding não interagente

As Equações de Movimento para o sistema não interagente são dadas por:

$$\boxed{(i\partial_t - \epsilon_i)G^r_{ij} = \delta(t-t')\delta_{ij} + t_{i+1,i}G_{i+1,j} + t_{i-1,i}G_{i-1,j}} \tag{7}$$

A dedução pode ser encontrada no Apêndice.

Note que agora o problema agora se resume a resolver um sistema de equações diferenciais acopladas e então fazer a Transformada de Fourier do sistema. Os aspectos numéricos e computacionais utilizados serão descritos abaixo.

Como descrito na seção anterior, a partir da EOM encontrada para as funções de Green, o problema agora está em resolver um sistema de Equações Diferenciais acopladas e então aplicar uma Transformada de Fourier ao resultado.

O sistema de EDs pode facilmente ser resolvido numericamente utilizando o Método de Runge-Kutta de 4ª ordem (RK4). A solução é da forma de uma soma de senos e cossenos de diferentes freqências em amplitudes.

Seguem os coeficientes usados no método RK4:

$$k_1^j = \frac{1}{i}\left(\epsilon_i G_{ij} + \delta(t)\delta_{ij} + \sum_{k=\pm1} t_{i+k,i}G_{i+k,j}\right)h$$

$$k_2 = \frac{1}{i}\left[\epsilon_i(G_{ij} + k_1^j/2) + \delta(t+h/2)\delta_{ij} + \sum_{k=\pm1} t_{i+k,i}(G_{i+k,j} + k_1^{j+k}/2)\right]h$$

$$k_3 = \frac{1}{i}\left[\epsilon_i(G_{ij} + k_2^j/2) + \delta(t+h/2)\delta_{ij} + \sum_{k=\pm1} t_{i+k,i}(G_{i+k,j} + k_2^{j+k}/2)\right]h$$

$$k_4 = \frac{1}{i}\left[\epsilon_i(G_{ij} + k_3^j) + \delta(t+h)\delta_{ij} + \sum_{k=\pm1} t_{i+k,i}(G_{i+k,j} + k_3^{j+k})\right]h$$

Em que h é o tamanho do passo usado na discretização do tempo.

Após resolver as funções de Green no tempo é necessário aplicar uma FT. Para este fim foi utilizada a função FFT do Numpy. Com isso resolvemos completamente o problema de encontrar a Densidade de Estados como função da energia.

Note que quanto maior o intervalo de tempo e o número de pontos, melhor o resultado da FFT. Assim, após resolver as Funções de Green numericamente para um período curto de tempo foram utilizadas Redes Neurais para aumentar o número de pontos e assim refinar o resultado.

2. Redes Neurais

2.1 Os Multi-Layered Perceptron (MLP)

O MLP foi um dos primeiros modelos de Redes Neurais Artificiais (ANN) a surgir e é até hoje o modelo mais popular, mas seu funcionamento evoluiu com o tempo, sendo muito otimizado.

A seguir é feita uma breve apresentação teórica sobre o funcionamento básico de um MLP e de alguns parâmetros importantes.

– *Representação dos Neurônios e Camadas e parâmetros importantes*

Essencialmente uma rede neural é um grafo acíclico(4) que pode assumir diversas formas e topologias. O modelo mais comum é o modelo Sequencial, cuja visualização pode ser vista abaixo.

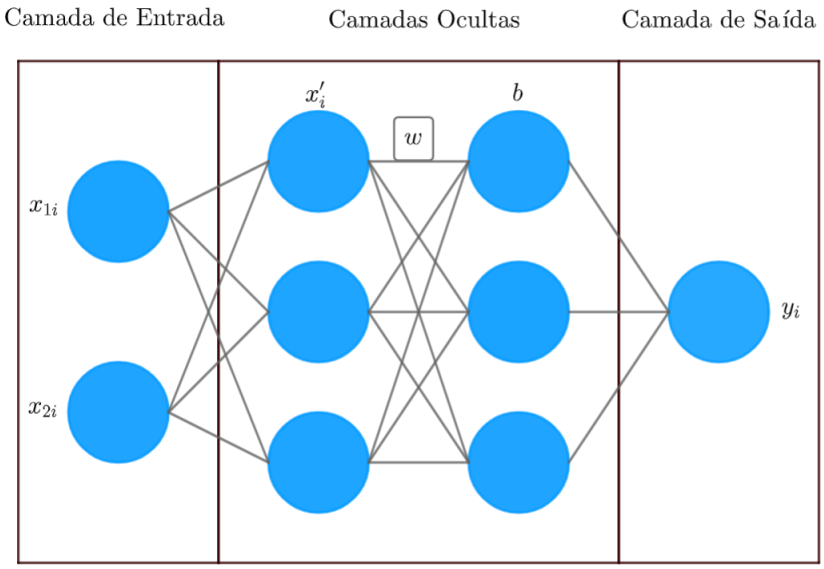


Figure 1: Representação visual de uma rede neural densa com duas entradas, uma saída e duas camadas ocultas de três neurônios cada. Note que esta representação se assemelha à de um grafo.

Os componentes básicos do MLP são seus parâmetros, w e b, que são os pesos e os bias, respectivamente. Essencialmente cada conexão entre neurônios tem um peso correspondente e cada neurônio possui um bias, de forma que, dado um vetor de entrada $X = (x_1, x_2, \cdot\cdot, x_n)$ o neurônio tem como saída

$$y = f\left(b + \sum_i w_i x_i\right)$$

Em que $f(x)$ é a chamada função de ativação. (link para funções de ativação)

Além desses parâmetros e da função de ativação (e de claro, o número de camadas e neurônios) um componente fundamental para uma rede neural, e que está diretamente ligado ao seu "aprendizado" é a função custo.

– *Como a rede neural "aprende" - Backpropagation*

Como mencionado na subsessão anterior, um fator fundamental para o aprendizado de uma ANN é a função custo. Para elucidar seu funcionamento, utilizaremos uma função custo bem conhecida: a MSE (Mean Squared Error):

$$C(\bar{Y}) = \frac{1}{N}\sum_i (y_i - \bar{y}_i)^2$$

Em que y_i é um output correto (estamos no contexto do Aprendizado Supervisionado) e $\bar{y}_i$ é um output dado pela NN. Note que $\bar{y}_i$ é uma função tanto das entradas como dos parâmetros definidos anteriormente (pesos e bias).

O mínimo da função custo pode ser encontrado via derivada

$$\vec{\nabla} C = 0$$

Em que as derivadas de C são tomadas por todos os parâmetros do sistema, de forma que é necessário o uso extensivo de regra da cadeia.

Dada a derivada da função custo em relação a um parâmetro w, por exemplo, podemos atualizá-lo usando

$$w^{(k+1)} = w^{(k)} - \eta \frac{\partial C}{\partial w}$$

Em que η é conhecido como taxa de aprendizado.

Esta técnica é conhecida como Gradiente Descendente, e é uma das formas de Backpropagation.

Na verdade, esta é uma forma simplista de enxergar como tudo ocorre. Na prática, a maioria dos algoritmos pré-prontos (como no Keras) existe uma opção mais eficiente que é o Gradiente Descendente Estocástico (SGD).

– *O problema das Séries Temporais*

Neste trabalho estamos interessados no uso de redes neurais especificamente no contexto de previsão de funções, e por isso trato isso como um problema de séries temporais.

Num problema de série temporais, basicamente temos uma lista com dados do "passado" e queremos predizer o futuro, e para isso esperamos que haja algum tipo de tendência e/ou padrão nestes dados, para que a NN consiga ser bem sucedida (que felizmente é o caso das GF).

De forma mais matemática, considere uma lista/sequência com n elementos:

$$X = (x_1, x_2, x_3, \cdots, x_n)$$

E com base nesta lista, queremos descobrir o elemento x_{n+1}.

2.2 *Redes Neurais Convolucionais (CNN)*

As CNNs foram e são amplamente utilizadas no mais diversos problemas, especialmente na análise de imagem e Visão Computacional. No entanto, é sabido que as CNNs são também úteis em problemas de séries temporais, especialmente quando há padrões claros nos dados.

Desta forma, se faz útil o estudo desta poderosa ferramenta.

– *Kernels e Convolução Discreta*

As redes convolucionais, como o nome diz, são baseadas nas operações de convolução e correlação cruzada discretas (já que computacionalmente os dados são inerentemente discretizados). Para exemplificar como ocorrem estas operações, considere duas matrizes A e k, em que A seria uma matriz de entrada (uma imagem, por exemplo) e k é o chamado Kernel. estas matrizes estão abaixo

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{pmatrix} \quad k = \begin{pmatrix} 1 & 2 \\ -4 & 0 \end{pmatrix}$$

Segue a operação de correlação cruzada (discreta):

$$A \star k = \begin{pmatrix} 1 \cdot (1+2+3+2) & 2 \cdot (2+3+2+1) \\ -4 \cdot (3+2+2+1) & 0 \cdot (2+1+1+3) \end{pmatrix}$$

$$\Rightarrow A \star k = \begin{pmatrix} 8 & 16 \\ -32 & 0 \end{pmatrix}$$

Na prática, essas convoluções condensam informação de forma a localizar padrões locais. Isso é muito útil para classificação de imagens, mas caso haja padrões explícitos numa série temporal, podemos utilizar a CNN, tomando a série como se fosse uma "imagem unidimensional".

– ***Operações e parâmetros importantes da CNN***

Abaixo estão listados e exemplificados algumas operações e parâmetros importante da CNN.

- Kernel size

  O kernel size, como o nome diz, é o tamanho/formato dos kernels.

- Stride

  O Stride informa ao algoritmo como vai ser a varredura do kernel sobre o input. Ele basicamente diz "de qaunto em quanto" o kernel vai "pular". Essa operação pode ser melhor elucidada com ilustrações. Seguem algumas abaixo.

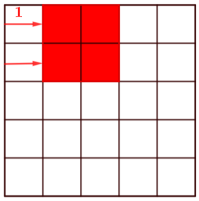


Figure 2: Representação gráfica da leitura de uma matriz por um kernel com stride (1, 1).

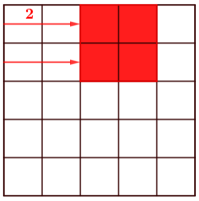


Figure 3: Representação gráfica da leitura de uma matriz por um kernel com stride (2, 2).

- Padding

  A operação de Padding consiste em adicionar linhas/colunas após o processo de convolução. No keras existem duas opções: "valid" e "same".

  Na opção "valid" nada é feito. Já na opção "same" são adicionadas linhas/colunas até que o formato pós-convolução seja igual ao formato pré-convolução. Veja uma ilustração.

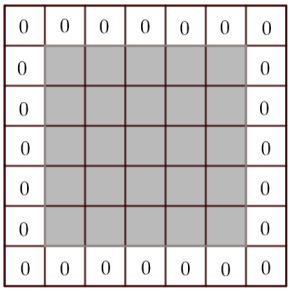


Figure 4: Representação gráfica do Padding.

- Pooling

  A operação de Pooling consiste em, dada uma matriz ou um vetor, "varrer" todo o conjunto de dados com uma janela de tamanho específico (pool size) de forma a reduzir e condensar os dados. Os principais tipos de Pooling são o Max Pooling e o Average Pooling, que dados os valores dentro da janela, eles retornam o valor máximo e o valor médio, respectivamente.

  Veja um exemplo de Max Pooling (pool size 2x2):

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \rightarrow \begin{pmatrix} 6 & 8 \\ 14 & 16 \end{pmatrix}$$

2.3 *Uma introdução ao TensorFlow-Keras no python*

O Keras(3)(4) é uma sub-biblioteca do Tensorflow usada especialmente para Deep Learning. Ele é extremamente eficiente, pois todas as operações por ele realizadas são compiladas em uma linguagem de nível mais baixo (usualmente C). Ele é de fácil utilização e é capaz de fazer redes neurais altamente customizáveis.

Para construir um modelo sequencial definimos o nome do modelo e então construímos camada por camada. Após arquitetada, precisamos compilar o modelo e então treiná-lo. Um modelo de rede densa está abaixo

```
#Importacao do modelo
from tensorflow import keras

#Inicializacao do modelo, com o nome modelo
modelo = keras.Sequential()

#Primeira camada. Nela e necessario informar o formato da entrada
#Aqui construimos uma camada de 32 neuronios e funcao de ativacao relu
modelo.add(keras.layers.Dense(32, activation = 'relu', input_shape = input_shape))

#Segunda camada, com 32 neuronios
#Desta vez a funcao de ativacao usada e a LeakyReLU
modelo.add(keras.layers.Dense(32))
modelo.add(keras.layers.LeakyReLU(alpha = 0.01))
```

```
#Camada de output, com um unico neuronio
#Para problemas de regressao/series temporais e comum nao utilizar funcao de
#ativacao nesta ultima camada
modelo.add(keras.layers.Dense(1))

#Compilacao do modelo.
#O otimizador utilizado e o SGD (Stochast Gradient Descent)
#A funcao custo e mse (mean squared error)
#E foi utilizada uma segunda metrica, mae (mean absolute error)
modelo.compile(optmizer = 'SGD', loss = 'mse', metrics = ['mae'])

#Treino da rede com os dados de treino
#Epochs corresponde ao numero de iteracoes realizadas no treino
#Batch_size corresponde ao numero de particoes que sao feitos no conjunto de dados
#Um batch_size grande faz um treino ser mais rapido,
#porem o modelo pode perder acuracia
#Ja um batch_size pequeno pode dar a maxima performance do modelo, mas faz o
#processo de treino ficar muito demorado
#Validation_split escolhe dados do conjunto de treino aleatoriamente
#e calcula a funcao custo
#Neste caso, o modelo utilizaria 20% dos dados para isso.
#Estes dados nao sao utilizados no treino
#Isso e importante para verificar se ha overfitting
#Note que para series temporais isso nao deve ser feito
#ja que a ordem dos dados importa
hist = modelo.fit(X_treino, y_treino, epochs = 10, batch_size = 20,
    validation_split = 0.2)

#Plot das funcoes custo e das metricas armazenadas na variavel hist
fig, ax = plt.subplots(1, 2)

ax[0].plot(hist.history['loss'], label = 'Funcao custo do treino')
ax[0].plot(hist.history['val_loss'], label = 'Funcao custo da validacao')
ax[0].legend()

ax[1].plot(hist.history['mae'], label = 'Metrica do treino')
ax[1].plot(hist.history['val_mae'], label = 'Metrica da validacao')
ax[1].legend()
```

Vistos os principais comandos do keras para uma rede simples, vamos para um exemplo de rede Convolucional:

```
modelo_conv = keras.Sequential()

#Primeira camada convolucional
#O numero de filtros seria equivalente ao numero de "neuronios"
#O input_shape normalmente espera um array de 3 a 4 dimensoes, da forma
#(batch_size, img_lenght, img_height, canais), mas nao e obrigatorio informar
#o batch_size
```

```
#Logo em seguida acionamos o comando para a operacao de max pooling
modelo_conv.add(keras.layers.Conv2D(filters = 32, kernel_size = (3, 1), activation
    = 'relu', input_shape = input_shape))
modelo_conv.add(keras.layers.MaxPooling2D(pool_size = (5, 1)))

modelo_conv.add(keras.layers.Conv2D(filters = 16, kernel_size = (3, 1), activation
    = 'relu'))
modelo_conv.add(keras.layers.MaxPooling2D(pool_size = (5, 1)))

#Como as camadas convolucionais retornam arrays de varias dimensoes,
#E as proximas camadas sao densas, que so aceitam arrays 1D
#Usamos o comando Flatten(), que faz um reshape da saida da camada convolucional
modelo_conv.add(keras.layers.Flatten())

modelo_conv.add(keras.layers.Dense(32, activation = 'relu'))

modelo_conv.add(keras.layers.Dense(1))

modelo_conv.compile(optmizer = 'SGD', loss = 'mse', metrics = ['mae'])

#Com o comando Summary() podemos analisar de forma mais visual a arquitura da rede
#E tambem obter o numero de parametros do sistema
modelo_conv.Summary()
```

Dado um modelo treinado, podemos fazer previsões usando o seguinte comando:

```
Modelo.predict(X_pred)
```

Como a utilização prática de redes neurais exige fazer "experimentos" alterando diversos parâmetros, como número de camadas, função de ativação, etc, é sempre necessário estudar a documentação da biblioteca.

3. Metodologia

Os cálculos numéricos consistem em duas etapas: a primeira, utilizando um métdp numérico convencional (RK4, neste caso), e uma segunda em que se faz uso de redes neurais.

A primeira etapa é importante pois, é nela que se retiram os dados que serão utilizados para treino e para teste/comparação. No caso do presente trabalho, faz-se o uso do Método de Runge-Kutta de 4a ordem para resolver o sistema de equações diferenciais representado pela equação (7), e cujos coeficientes estão expostos na seção 1.4.

Com estes dados em mãos, lançamos mão do uso de redes neurais para resolver um problema de séries temporais.

Simplificadamente, podemos trabalhar com o chamado **single-step forecast**, que consiste em, dada uma lista de n elementos ordenados,

$$(x_1, \cdots, x_n)$$

Prever o elemento x_{n+1}.

Como, em geral, não queremos calcular apenas um ponto, devemos fazer um **multi-step forecast**. o método mais simples, que foi utilizado aqui, é um método recursivo, que consiste em tomar uma lista de n elementos, uma janela de comprimento m, e prever elementos sucessivamente da seguinte forma:

$$X^{(0)} = (x_1, \cdots, x_n)$$
$$X^{(1)} = (x_1, \cdots, x_n, x_{n+1})$$
$$X^{(2)} = (x_2, \cdots, x_n, x_{n+1}, x_{n+2})$$
$$\cdots$$

É claro que este método apresenta erros cumulativos, e por isso faz-se necessário um estudo mais aprofundados de outros métodos de previsão de múltiplos passos, que será feito ainda no decorrer do projeto.

4. Resultados

Utilizando a equação (7), e mais especificamente os coeficientes explicitados na sessão 1.4 podemos calcular a GF em função do tempo. Após resolver as equações numericamente e separar em partes real e imaginária, separamos os dados que serão usados para treino e os de teste.

Abaixo podemos ver as partes real e imaginária da função de Green para um dos sítios, numa rede de 11 sítios. O gráfico em azul demarca a solução exata/numérica usando RK4 e a linha pontilhada em vermelho é o resultado da CNN. linha verde demarca onde acaba a região de treino e começa a região de teste e previsão da rede.

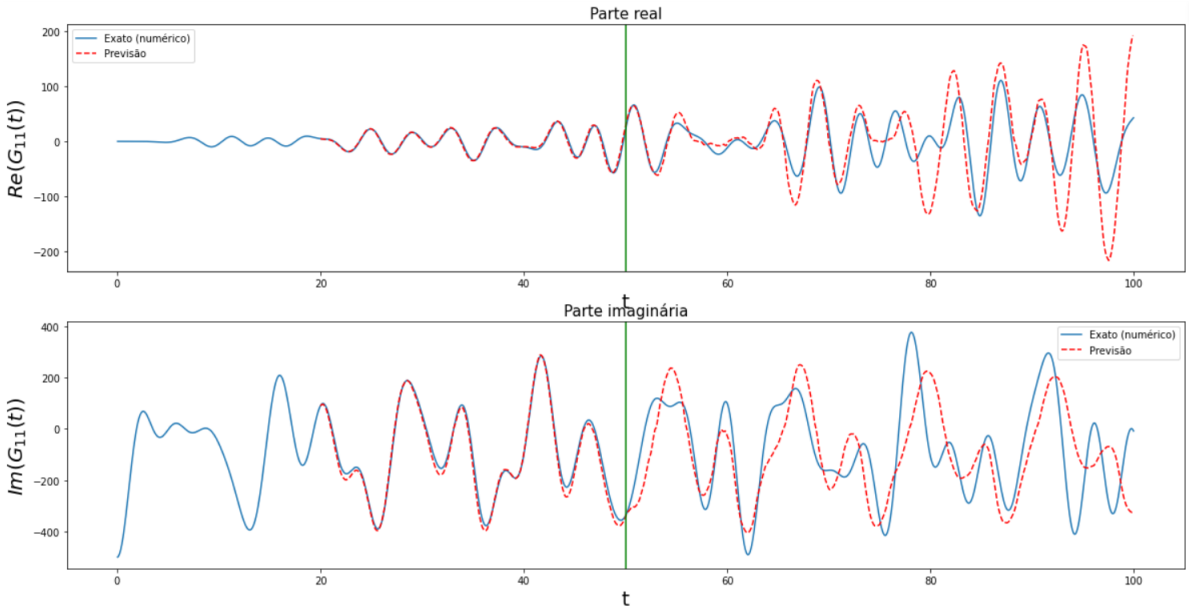


Figure 5: Partes real (acima) e imaginária (abaixo) da Função de Green em função do tempo para o sítio 1 de uma cadeia unidimensional de 11 sítios não interagentes. Como mostra a legenda, a curva azul representa o resultado "exato" (numérico) e a curva vermelha pontilhada representa a extrapolação da rede neural a partir da linha verde, que onde acabam os dados de treino.

Com a GF em função do tempo, tomamos a Transformada de Fourier (usando FFT) para trabalharmos no domínio da frequência. Utilizando então a equação (5) podemos encontrar a densidade de estados.

Abaixo estão dois gráficos da densidade de estados em função da frequência, um para a solução numérica via RK4 e o de baixo corresponde ao resultado da CNN.

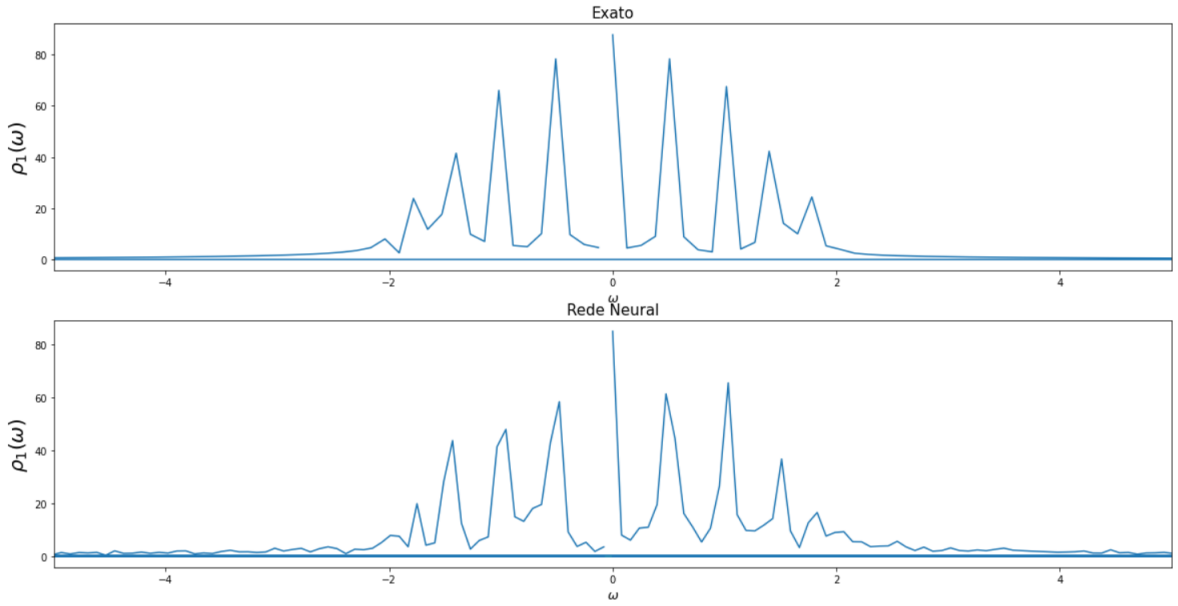


Figure 6: Transformada de Fourier das t-GF apresentadas na fígura anterior. Acima está o resultado para a solução "exata" e abaixo está o resultado para a rede neural. Note que a NN consegue reproduzir com exatidão as frequências em que ocorrem os picos da densidade de estado.

Note que a CNN conseguiu reproduzir quase que perfeitamente a densidade de estados do caso numérico, apesar de apresentar algum ruído.

No mais, um fato preocupante está relacionado aos tempos de processamento para o cálculo das funções de Green: via métodos numéricos convencionais (RK4) o tempo de processamento foi de 107s, enquanto que para a NN, considerando treino e previsão, o tempo total foi de 1200s (ambos calculando a função de Green até T = 100). Portanto, parece que para este tipo de problema em específico os métodos convencionais são mais eficientes, sendo cerca de 10x mais rápidos e tendo uma precisão ligeiramente maior.

4.1 Comparação com Kwant

Para fins de comparação, foi utilizada a biblioteca Kwant(2), no python, para calcular a Função Espectral do modelo estudado. Seguem as funções espectral obtidas pelo Kwant, por Funções de Green (numérico/RK4) e pela NN, respectivamente.

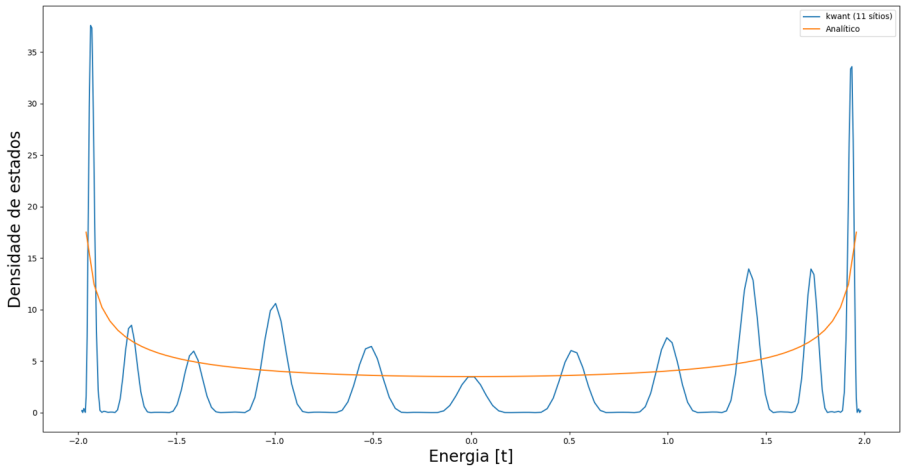


Figure 7: Função espectral para uma cadeia unidimensional de 11 sítios. Em azul temos o resultado do Kwant e em laranja temos um resultado analítico obtido tomando $N \to \infty$.

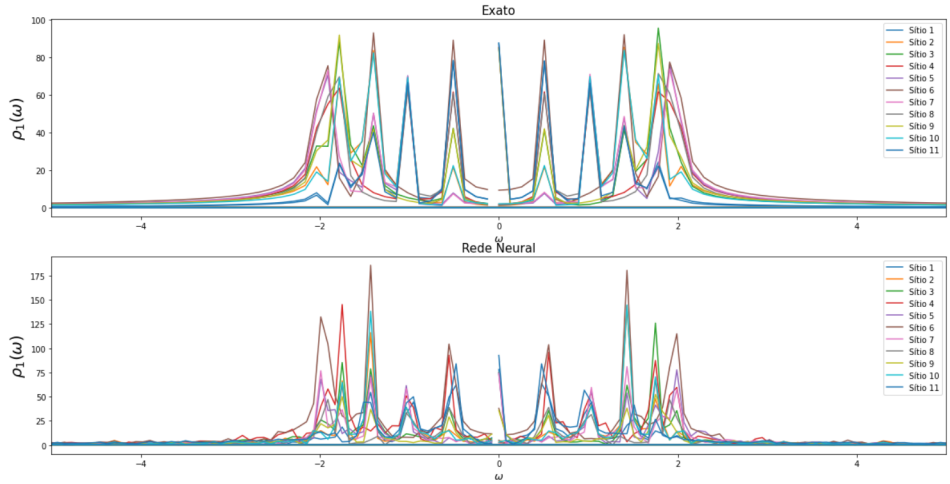


Figure 8: Funções espectrais para todos os sítios da cadeia unidimensional de 11 sítios. Acima está o resultado obtido através da simulação com RK4 e abaixo está o resultado obtido via NN.

5. Conclusão

Como visto na sessão de Resultados, para o caso de sistemas não interagentes não é prático e viável utilizar redes neurais. Tal fato era esperado, já que a complexidade da maioria dos algoritmos de solução numérica de equações diferenciais ordinárias (i.e RK4 e Verlet) é $O(kn)$, em que n é o número de equações (que é igual ao número de partículas) e k é o número de passos no tempo. Enquanto isso, as redes neurais tem complexidade $O(nki)$, em que n é o número médio de neurônios por camada, k é o número de camadas e i é a quantidade de outputs, que neste caso seria o número de passos no tempo.

Além disso, é necessário um estudo de caso do modelo e dos dados, já que o método de extrapolação utilizado é o chamado **Recursive Single-Step Forecast**, que resulta em erros cumulativos. Uma solução seria o uso de **Direct Multi-Step Forecast** ou um modelo híbrido do

primeiro com o segundo. No entanto, estes métodos exigem uma quantidade grande de dados de treino, e por isso deve-se avaliar a viabilidade em cada caso.

Desta forma, o uso de NNs é inviável para este caso simples. No entanto, sabemos que para problemas interagentes, em que se faz necessário o uso de DMRG, por exemplo, a complexidade é $O(2^{3N})$ em que N é o número de partículas, de forma que o uso de ANNs deve se mostrar viável.

6. Planos futuros

Os próximos passos são:

- Estudar o uso conjunto de RNNs e CNNs para a predição de séries temporais;
- Estudar o uso de outros tipos de topologia das NNs além do modelo Sequencial;
- Estudar o modelo de Impureza de Anderson e implementar sua solução com base em t-DMRG utilizando o pacote ITensor(5) no C++;
- Explorar o uso das Redes Neurais para a otimização da solução do modelo mencionado acima.

7. Apêndice

Neste apêndice estão disponibilizados códigos e cálculos relacionados a este projeto.

- Dedução da Equação de Movimento para as GF não interagentes (Link);
- Código das redes neurais (Link).

$$G^R_{ii}(t,t') = -i\theta(t-t')\langle\{c_i(t),c_i^\dagger(t')\}\rangle$$

References

[1] Antoine Honet, Luc Henrard, Vincent Meunier; Exact and many-body perturbation solutions of the Hubbard model applied to linear chains. AIP Advances 1 March 2022; 12 (3): 035238.

[2] Kwant. Calculating spectral density with the kernel polynomial method. https://kwant-project.org/doc/dev/tutorial/kpm

[3] Keras. Keras. https://keras.io/

[4] Chollet F. Deep Learning with Python.

[5] Itensor. Itensor. https://itensor.org/docs.cgi