Using Neural Networks to Solve Schrodinger's Equation : Particle in a Box

Anwesh Bhattacharya

March 21, 2020

1 Particle in a Box

Time independent Schrodinger's equation is as follows -

$$\frac{-\hbar^2}{2m}\frac{\mathrm{d}^2\psi}{\mathrm{d}x^2} + V(x)\psi(x) = E\psi(x)$$

The potential for the system is as follows (*L is the length of the box*) -

$$V(x) = \begin{cases} 0 & ;\ 0 \leq x \leq L \\ \infty & ;\ \text{otherwise} \end{cases}$$

It is known that the infinite potential kills the wavefunction to 0. Hence, focusing only on the region $0 \leq x \leq L$, we get -

$$\frac{-\hbar^2}{2m}\frac{\mathrm{d}^2\psi}{\mathrm{d}x^2} = E\psi(x)$$

For this system, after applying boundary conditions and normalisation constraint, the analytical solution is -

$$\psi(x) = \sqrt{\frac{2}{L}} sin(\frac{n\pi x}{L})$$

where the eigenvalue E is

$$E_n = \frac{n^2\hbar^2\pi^2}{2mL^2}$$

2 Basic Network and Loss Setup

NeuroDiffEq can handle second-order differential equations with boundary conditions. The differential equation, in its functional form, is posed as an MSE loss function which the neural network optimizes. However in the case of Schrodinger's equation, the differential is not determined upto its eigenvalue E. To the best of my knowledge, the existing framework could not be used simply to solve Schrodinger's equation. Hence, the network needs an extra parameter E, that also has to be learnt over the training process.

Consider a general two-point boundary condition which is $\psi(x_0) = \psi_0$ and $\psi(x_1) = \psi_1$. If the NN output is ψ_N, then the boundary condition is enforced as follows (*as done in NeuroDiffEq*) -

$$\tilde{\psi} = \psi_0(1-\tilde{x}) + \psi_1\tilde{x} + (1 - e^{(1-\tilde{x})\tilde{x}})\psi_N \quad \text{where} \quad \tilde{x} = \frac{x - x_0}{x_1 - x_0}$$

Considering $\hbar = m = 1$ and denoting $p = \int_{-\infty}^{\infty} |\tilde{\psi}(x)|^2 dx$, the loss function becomes -

$$\mathit{Loss} = \left(-\frac{1}{2}\frac{\mathrm{d}^2\tilde{\psi}}{\mathrm{d}x^2} - E\tilde{\psi}\right) + (1-p)^2$$

This loss function too enjoys the unique property that it is identically zero at the correct (*normalized*) solution. In principle, there are an infinite number of eigenvalues and eigenfunctions. The neural network is sensitive to the initial value of the E parameter. It will try to find an eigenvalue close to the starting value.

The notebook was built upon the code written by the *Kitchin* group at the Chemical Engineering Department as CMU here. 2 hidden layers of size 32 each with *tanh* activation were used. Their code was written to solve the quantum harmonic oscillator using *autograd* library, however it has been modified according to the framework in *NeuroDiffEq* .

2.1 n = 1, L = 1

The eigenvalue for this case is $\frac{\pi^2}{2} = 4.9348022$. Initialising E in the range $[4, 6]$, it is converging to the correct value within 5000 epochs and the plot is also correct (*except that it could be inverted about the x-axis as the normalisation constraint is sign invariant*). The following is the graph obtain for $E = 5.5$.

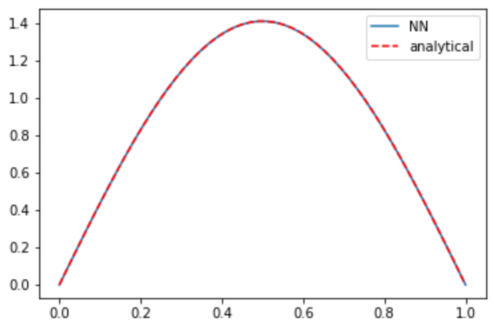

Figure 1: Wavefunction for n=1, L=1

It is matching very well with the analytical curve. This is marked by the final value of the MSE loss - 0.01044 - and the learned eigenvalue - 4.93483818 - which is accurate upto 4 places of decimal. The MSE loss can be further decreased to the order of 10^{-5} or lesser with 10,000 training epochs.

It is informative to check some diagnostic curves for this case. The *MSE*vs. *iterations* curve is below -

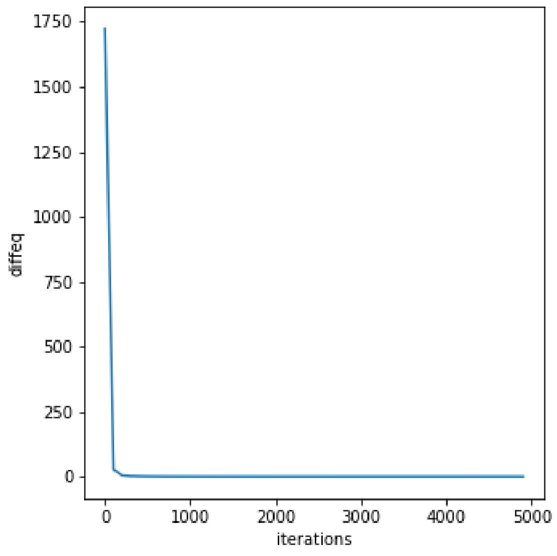

Figure 2: MSE curve for n=1, L=1

The MSE loss quickly falls to zero in a few epochs. It is also helpful to observe the behaviour of other network parameters in diagnostic plots. The other two curves plotted below are that of *probability* vs. *iterations* and *eigenvalue* vs. *iterations* -

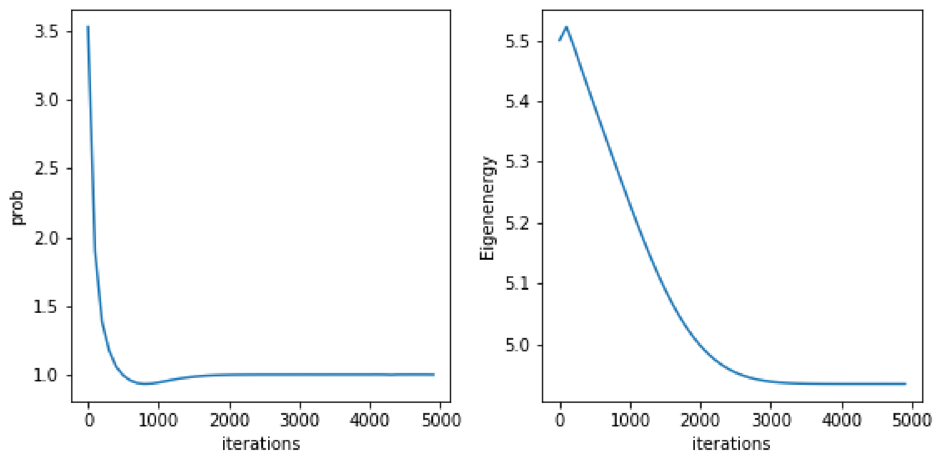

Figure 3: Probability & Eigenvalue curve for n=1, L=1

The probability needed a couple thousand of epochs to begin settling to 1. The eigenvalue also flattens to its correct value in a similar fashion.

2.2 n=2, L=1

The correct eigenvalue is $2\pi^2 = 19.7392088$. Initialising the energy parameter E to 22, the following plot is obtained -

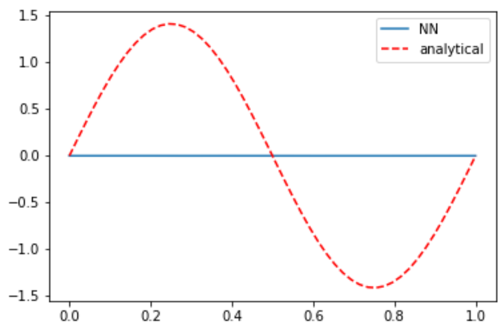

Figure 4: Flat Wavefunction for n=2, L=1

The network has learned a flat wavefunction. Although the MSE term is low (0.0022), the final probability obtained is vanishingly low at 1.4×10^{-7}. The eigenvalue learned is also incorrect at 22.967. The reason for this can gauged from the *probability* vs. *iterations* diagnostic curve below

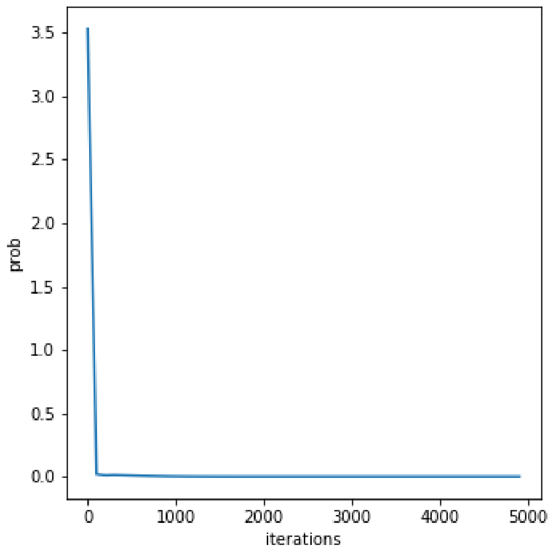

Figure 5: Probability curve for n=2, L=1

It isn't the case that the probability slowly falls to zero over the iterations. The probability plummets to zero immediately, and *also* results in a flat wavefunction. In terms of optimization of the loss function, this makes sense as a flat wavefunction is a *trivial* way to make the *MSE* term vanish. A similar behaviour was noticed with other cases of n and L (*especially for higher values of the ratio* $\frac{n}{L}$) and concluded that the network has a tendency to flatten the wavefunction.

3 Probability Regularization

A possible solution is to avoid the plummeting of the probability is to strengthen the loss function with regularization terms as function of probability p. Its purpose is two-fold -

1. It would prevent the probability from hitting zero
2. It would force the network to learn a *non-trivial* eigenfunction and the correct eigenvalue

A regularisation term of $\frac{1}{p}$ would force the probability away from 0, and there is no harm in doing so as $p \geq 0$ by construction. However, this would have the side-effect of pushing the probability beyond 1 as $\frac{1}{p}$ is a monotonically decreasing function of p. This can be counter-acted by linear or higher degree terms such as p or p^2.

As a generic case, the regularization function was chosen as follows -

$$R(p) = \frac{a}{p} + \frac{c}{p^2} + \frac{e}{p^3} + \frac{g}{p^4} + bp + dp^2 + fp^3 + hp^4$$

It needs to be ensured that the loss function has a local minima at $p = 1$. The $(1-p)^2$ term already has a global minima at $p = 1$. Hence, finding the derivative of R with respect to p is sufficient -

$$\frac{\mathrm{d}R}{\mathrm{d}p} = -\frac{a}{p^2} - \frac{2c}{p^3} - \frac{3e}{p^4} - \frac{4g}{p^5} + b + 2dp + 3fp^2 + 4hp^3$$

Setting this derivative 0 at $p = 1$, we get -

$$a + 2b + 3c + 4g = d + 2e + 3f + 4h$$

It would be desired that in the neighbourhood of $p = 1$, the dominating regularising terms are $\frac{a}{p}$ and dp and not the other terms. The higher exponent terms will automatically dominate for a relatively larger value of $|p-1|$. Hence, we must ensure that Thus, one can choose -

$$c = \frac{a}{4}\ ,\ e = \frac{c}{6}\ ,\ g = \frac{e}{8}$$

and

$$d = \frac{b}{2}\ ,\ f = \frac{d}{6}\ ,\ h = \frac{f}{8}$$

If we denote Δp_a as the change in the probability due to the $\frac{a}{p}$ term at $p = 1$, and likewise for other terms, then the choice of co-efficients above ensures that -

$$\Delta p_a = 2\Delta p_c = 4\Delta p_e = 8\Delta_g\ ,\ \Delta p_b = 2\Delta p_d = 4\Delta p_f = 8\Delta p_h$$

Making an arbitrary choice of $a = d = 20$, the final regularisation function is -

$$R = \frac{20}{p} + \frac{5}{p^2} + \frac{5}{6p^3} + \frac{5}{48p^3} + 20p + 5p^2 + \frac{5p^3}{6} + \frac{5p^4}{48}$$

Note that the condition $a + 2b + 3c + 4g = d + 2e + 3f + 4h$ is automatically satisfied by construction.

Now the minimum value the loss function could attain is $(a + b + c + d + e + f + g + h)$ which turns out to be 51.875 in our case. The MSE term could still attain a lowest possible value of 0, however. The plot of only the p terms of the loss $L(p) = (1-p)^2 + R(p)$, and its derivative, is given below -

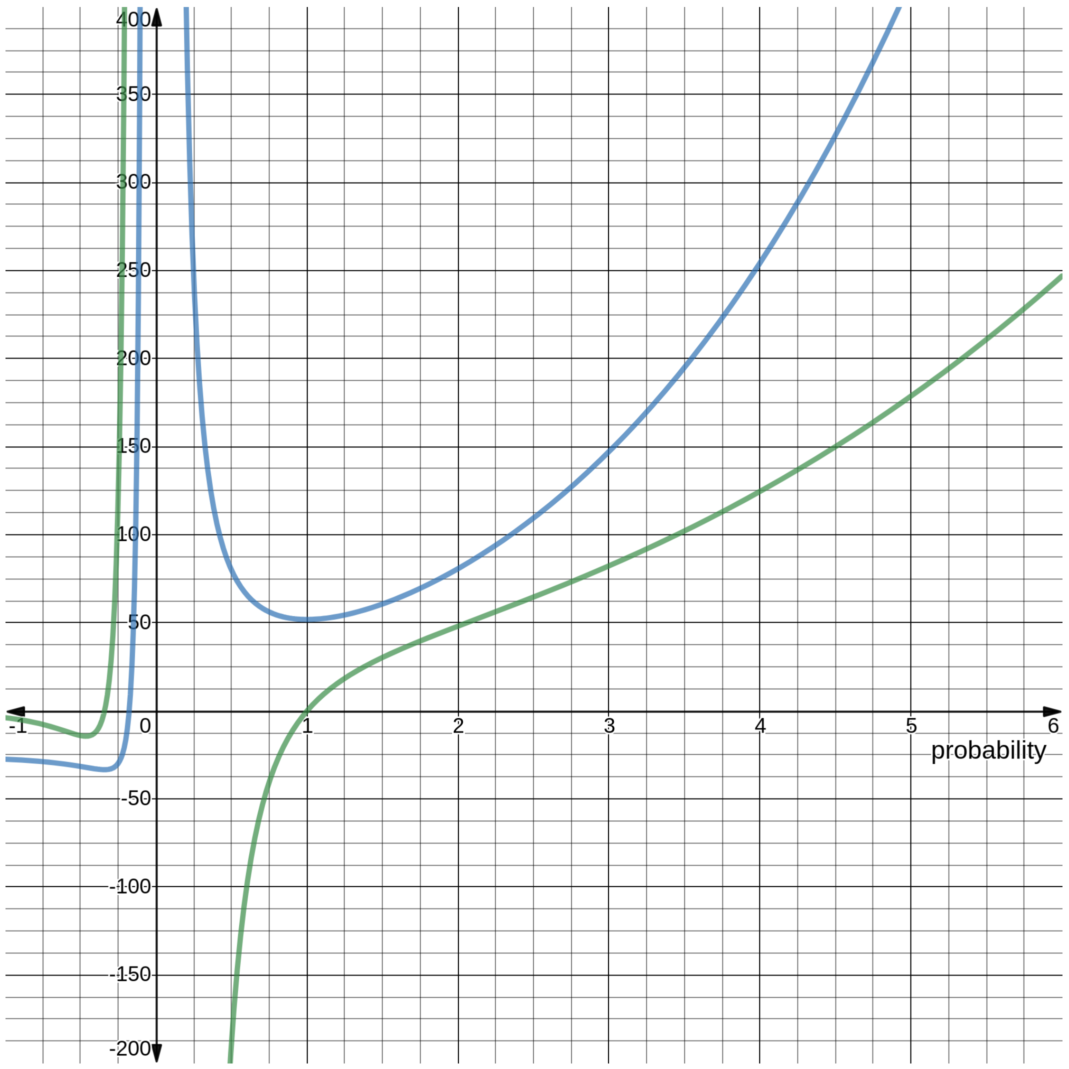

Figure 6: Probability Regularization graph

(a) Blue : Regularization term, Green : Its derivative

There is a strong derivative for $p < 1$ to prevent the probability from falling to 0. At the same time, there is also a moderately strong derivative for $p > 1$ which prevents it from overshooting.

3.1 n=2, L=1

The above scheme of regularisation works extremely well for this case. The final value of the MSE loss is 0.00108643, and the final probability is 1.0000119. The eigenvalue learned is 19.73925203 which is again correct upto 4 places of decimal.

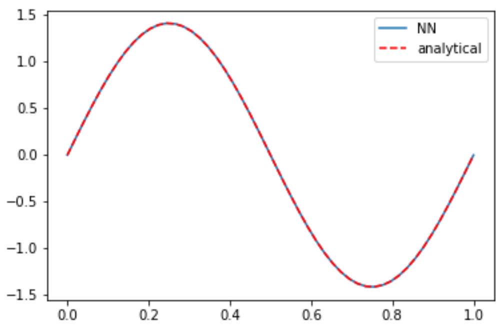

Figure 8: Wavefunction for n=2, L=1

The neural network learns the desired wavefunction. It is also interesting to look at the *probability* vs. *iterations* curve to check the effect of the regularization terms -

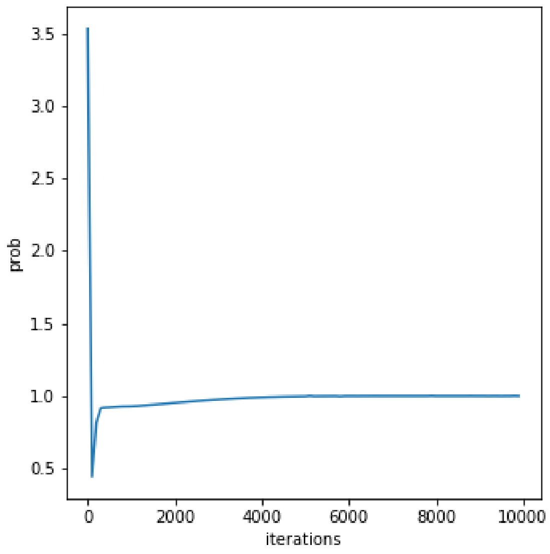

Figure 9: Regularized Probability curve for n=2, L=1

The initial behaviour of the probability is the same as in the previous case, however the regularization terms push the probability away from 0. It then grows upto 1 quickly and oscillates around it over the iterations. In general it was observed, that for lower values of the ratio $\frac{n}{L}$, the regularisation works quickly. For higher values of the ratio, the probability flattens out to 1 slower. Hence, the number of training iterations cannot be fixed for every case and this very issue is discussed in the next section.

3.2 n = 5, L = 2

The true eigenvalue is $\frac{5^2\pi^2}{22^2} = 30.8425137$ and the learned value is 30.84252368 which is again correct upto 4 decimal places. The probability obtained is 0.997989. The final MSE loss is 0.16477485. This is higher than the previously obtained losses by 2 orders of magnitude, but this could be explained by two factors -

1. The wavefunction for n=5 has 5 extremas and a continuously changing curvature which is a source of error in the $\frac{\mathrm{d}^2\tilde{\psi}}{\mathrm{d}x^2}$ term of the loss function
2. Limited resolution of the training set over the range $[0, L]$, coupled with the normalization constraint introduces error in the probability density numerical integration.

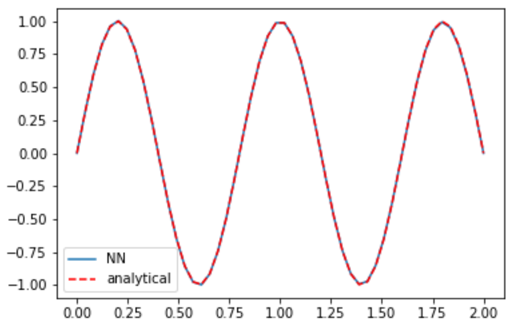

Figure 10: Wavefunction for for n=5, L=2

The diagnostic curves of *probability* vs. *iterations* and *eigenvalue* vs. *iterations* are well-behaved too, although they stabilize slower than the $n = 1, L = 1$ case. They flatten to their stable values at approximately the same number of epochs simultaneously. It is desired that this simultaneous stabilization is observed across all cases as it would imply that the eigenfunction and eigenvalue enforce each other strongly through the derivatives of the loss. It is also desired that there is significant flattening over the last couple thousand epochs of the training as it would allow the eigenfunction and eigenvalue to obtain precision.

In other cases, especially for higher ratios of $\frac{n}{L}$, the simultaneity of the probability and eigenvalue is not achieved strongly over the epochs. The probability reaches 1 too quickly and the eigenvalue fails to be trained, or it occurs that the eigenvalue is nearly accurate but the growth rate of the probability is too slow.

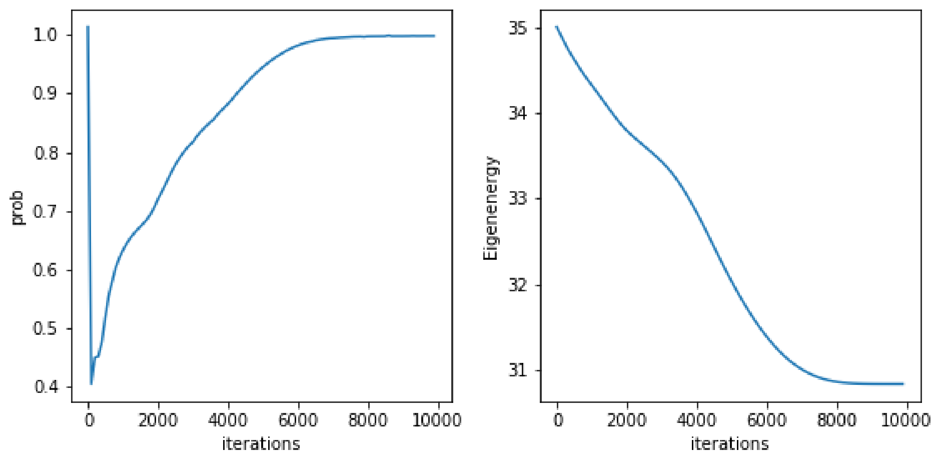

Figure 11: Probability and Eigenvalue curves for n=5, L=2

4 Exit Conditions

It is interesting to observe the case of $n = 7, L = 1$. This has a high ratio of $\frac{n}{L}$ compared to any case seen previously. After training for 150,000 epochs, a loss of 0.04669563 is obtained which is an order of magnitude higher than the previous cases, coupled with the large number of epochs. The wavefunction graph also has large granularity due to the limit of resolution on the real number line, which chokes the precision of all loss terms greatly. Also, the final probability is 0.99331093 which only confirms this fact, and is an order magnitude lower in precision compared to previous cases. The eigenvalue, however, was learnt accurately as 241.80534101 which is again precise upto 4 decimal places in comparison to $\frac{49\pi^2}{2} = 241.8053078$

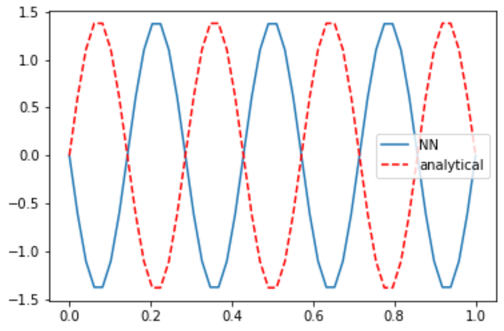

Figure 12: Wavefunction for n=7, L=1

It is natural to quit training once a desired level of precision is reached in the loss. The loss, in our case, has too many terms and hence its behaviour isn't reliable enough upon which to build an exit condition. Observing the behaviour of the MSE, probability, and eigenvalue terms over the training of many cases, the following observation could be made upon an exit condition to stop training -

1. It is not sensible to fix the number of epochs to an extremely high number. For cases with small valueof $\frac{n}{L}$, less than 10,000 epochs are sufficient to reach precisions of 10^{-6} in the MSE. Although there is no fear of *overfitting* in this Machine Learning problem, it would lead to wastage of time.
2. It wouldn't be advisable to increase the learning rate of the underlying optimization algorithm. The loss function is dotted with multiple local minima due to the existence of multiple eigenvalues.
3. The exit condition cannot be depend upon the precision reached on the probability term (*such as* 10^{-3} *or* 10^{-4}) due to the limit of resolution on the training set.
4. It cannot depend upon the precision reached on the loss due to similar reasons, and for wavefunctions with continuously changing curvature - higher ratios of $\frac{n}{L}$. In some cases, the jump from 10^{-2} to 10^{-3} could take $20,000$ epochs, whereas the jump from 10^{-3} to 10^{-4} could take more than $100,000$
5. Neither can it depend on the precision of the eigenvalue as it is a matter of scale of the problem, which we cannot judge *a priori*
6. On the other hand, the *difference* in the MSE every hundred epochs ,ΔMSE_{100} , has proven to have reliable behaviour across all cases. At the end of training it has taken values in the order of 10^{-4} Logically, it is sensible to use this to decide the halting of the training due to the following reasons
   - ΔMSE_{100} could reach a precision of 10^{-4} even at lower values of probability such as 0.98 or 0.95. It is not necessary that very high levels of precision in probability be reached before ΔMSE_{100} could enjoy high precision. Hence, it would take care of cases of high curvature and limited numerical resolution.
   - When ΔMSE_{100} reaches a certain level of precision, it means that the limit of numerical resolution of the system has been reached. It would avoid any pitfalls created by point 1 above. In other words, it wouldn't matter if the MSE saturated at order of magnitude 10^{-1} or 10^{-3}. If the resolution is choked, the ΔMSE_{100} can only improves at a much lower precision, which has been chosen as 10^{-4}
   - Exit conditions based on ΔMSE_{100} would handle cases where the training does not exhibit the simultaneous stabilization of the probability and the eigenvalue.

Having enlisted these observations, the **`adam`** optimizer was modified with the use of **`callback`** to take into account the exit conditions. The epoch at which ΔMSE_{100} reaches a value of 10^{-2} denotes the onset of precision of the network. This point could be reached before or after 10,000 epochs, which has been taken as a heuristic cutoff. Based on either case, two exit conditions have been designed - **`L3TExit`** (*less than ten thousand*) and **`NormalExit`** , respectively . Also depending on the subsequent behaviour of the training, **`L3TExit`** could transition to **`NormalExit`** , however **`NormalExit`** doesn't transition when it's set at the onset of precision.

4.1 Epoch Limits During Training

As the training loop goes through its iterations, various epoch limits are set based on the precision of ΔMSE_{100} .

Algorithm 1 **`Train_setEpchLim`**

```
 1: /* $i_2 \leftarrow$ Epoch number at which $\Delta\text{MSE}_{100}$ hits $10^{-2}$ */
 2: /* $lim_3 \leftarrow$ Epoch limit before which $\Delta\text{MSE}_{100}$ should hit $10^{-3}$ */
 3: /* $i_3 \leftarrow$ Epoch number at which $\Delta\text{MSE}_{100}$ hits $10^{-3}$ */
 4: /* $lim_4 \leftarrow$ Epoch limit before which $\Delta\text{MSE}_{100}$ should hit $10^{-3}$ */
 5: /* $i_4 \leftarrow$ Epoch number at which $\Delta\text{MSE}_{100}$ hits $10^{-4}$ */
 6: /* $endlim \leftarrow$ Epoch limit before which $\Delta\text{MSE}_{100}$ should hit $10^{-4}$ 5 times */
 7: /* $count \leftarrow$ No. of times $\Delta\text{MSE}_{100}$ has hit $10^{-4}$ or below */
 8: Initialise $i_2, lim_3, i_3, lim_4, i_4, endlim$ to $NULL$ if not already set
 9: Initialise $count, epch$ to 0
10: while True do
11:     $epch \leftarrow epch + 1$
12:     /* Perform training */
13:     if $\Delta\text{MSE}_{100}$ hits $10^{-2}$ then
14:         $i_2 \leftarrow$ epch
15:         $lim_3 \leftarrow 2i_2$
16:     if $\Delta\text{MSE}_{100}$ hits $10^{-3}$ then
17:         $i_3 \leftarrow epch$
18:         if $i_3 + 4(i_3 - i_2) > 2i_3$ then
19:             $lim_4 \leftarrow 2i_3$
```

Algorithm 2 **`Train_setEpchLim`** (Continued)

```
20:         else if $\left\lceil \frac{3i_3}{2} \right\rceil < i_3 + 4(i_3 - i_2) < 2i_3$ then
21:             $lim_4 \leftarrow i_3 + 4(i_3 - i_2)$
22:         else
23:             $lim_4 \leftarrow \left\lceil \frac{3i_3}{2} \right\rceil$
24:         if $lim_4 < lim_3$ then
25:             $lim_4 \leftarrow 2lim_3 - lim_4$
26:     if $\Delta\text{MSE}_{100}$ hits $10^{-4}$ then
27:         $count \leftarrow count + 1$
28:         $i_4 \leftarrow epch$
29:         if $i_4 + 8(i_3 - i_2) > 2i_4$ then
30:             $endlim \leftarrow 2i_4$
31:         else if $\left\lceil \frac{3i_4}{2} \right\rceil < i_4 + 8(i_4 - i_3) < 2i_4$ then
32:             $endlim \leftarrow i_4 + 8(i_4 - i_3)$
33:         else
34:             $endlim \leftarrow \left\lceil \frac{3i_4}{2} \right\rceil$
35:         if $endlim < lim_4$ then
36:             $endlim \leftarrow 2lim_4 - endlim$
37:     if $i2 > 10,000$ then
38:         call NormalExit
39:     else
40:         call L3TExit
```

4.2 `NormalExit`

This exit condition will be used when ΔMSE_{100} reaches 10^{-2} for the first time after 10,000 iterations. In this case, the training will exit when ΔMSE_{100} holds a value of 10^{-4} atleast 5 times, or any of the three limiting epochs are reached.

Algorithm 3 **`NormalExit`**

```
1: if $i_4$ is not $NULL$ then
2:     if ($count < 5$ and $epch > endlim$) or $count \geq 5$ then
3:         break
```

Algorithm 4 **NormalExit** (Continued)

```
4: if $i_3$ is not $NULL$ and $epch > lim_4$ then
5:     break
6: if $i_2$ is not $NULL$ and $epch > lim_3$ then
7:     break
```

4.3 `L3TExit`

In this case, the training will exit when ΔMSE_{100} holds a value of 10^{-4} atleast 10 times, or it decides to switch to **`NormalExit`** after checking any of the limiting epochs with a threshold of $20,000$.

Algorithm 5 **L3TExit**

```
1: if $i_3$ is not $NULL$ and $lim_4 > 20,000$ then
2:     call NormalExit
3: if $i_4$ is not $NULL$ then
4:     if $endlim > 20,000$ then
5:         call NormalExit
6:     else if $count >= 10$ then
7:         break
```

4.4 n = 7, L = 1

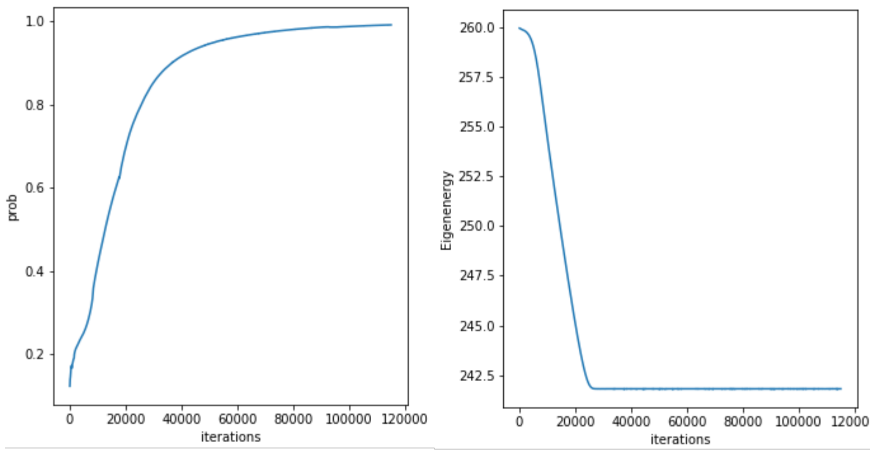

Figure 13: Wavefunction for n=7, L=1

When **adam** with the exit conditions was run on this case, it needed a total of 111,000 epochs. Thus, 40,000 epochs were saved from the previous run with negligible difference in precision. The eigenvalue reported as the same precision, and the loss is 0.043613 which is infact slightly better than the previous run with 150,000 epochs. The probability is also 0.990647.

The exit conditions have allowed the probability enough epochs to settle whilst not overtraining the network. It is also interesting to observe how the eigenvalue settles around epoch 25,000 whereas the probability reaches precisions of 10^{-1} only above epoch 100,000. This is a clear sign of the limited numerical resolution of the system.

5 Eigenvalue Regularization

The only control over the energy parameter is its initialization. The user of the network has no say over the eigenvalue the network will learn, as there are many of them. It would be desirable to restrict the search of the eigenvalue within a certain range. This could be achieved, again, by regularization terms.

5.1 Derivation

The range of eigenvalues $[E_1, E_2]$ to search in defines the scale of the problem. It can easily be captured by the number $m = \log E_2 - E_1$ (*to the base 10*). Consider a regularization function $f(E)$ -

$$f(E) = \frac{1}{k}\left[\frac{1}{(E-r_1)^n} + \frac{1}{(E-r_2)^n}\right]$$

Its derivative is -

$$f'(E) = -\frac{n}{k}\left[\frac{1}{(E-r_1)^{n+1}} + \frac{1}{(E-r_2)^{n+1}}\right]$$

r_1 and r_2 define singularity points and asserting that $E_1 = r_1 + \epsilon_1$ and $E_2 = r_2 - \epsilon_2$, where the neural network would search the energy parameter. Since the learning rate of **adam** is 10^{-3}, we can set the following -

$$10^{-3}\,|f'(E_1)| < 10^{m-4} \quad \text{and} \quad 10^{-3}\,|f'(E_2)| < 10^{m-4}$$

When the derivative is fixed in such manner, it ensures that the contribution to ΔE due to regularization is small in the middle of the range $[E_1, E_2]$, where the neural network is expected to find a stable eigenvalue. It's undesirable for this stability to be disturbed due to the above terms. It is also necessary to have a strong derivative near the singularities to contain the eigenvalue within the range. Using the first condition, we get -

$$\left| -\frac{n}{k}\left[\frac{1}{(\epsilon_1)^{n+1}} + \frac{1}{(r_1 - r_2 + \epsilon_1)^{n+1}}\right]\right| < 10^{m-1}$$

which gives -

$$\left[\frac{1}{(\epsilon_1)^{n+1}} + \frac{1}{(r_1 - r_2 + \epsilon_1)^{n+1}}\right] < \frac{10^{m-1}k}{n}$$

The second term on the LHS can be ignored as $|r_1 - r_2| >> \epsilon_1$. In particular, we can choose $n = 8$ and state that $n \sim 10^1$ which gives -

$$\epsilon_1^{-(n+1)} < 10^{m-2}k \implies \epsilon_1 < \left(10^{m-2}k\right)^{-\frac{1}{n+1}}$$

Since ϵ_1 is the distance between an eigenvalue limit and the singularity, we can choose that $\epsilon_1 < 10^{m-2}$. Taking k of the form 10^s, we can equate the RHS of the above two inequalities -

$$\left(10^{m-2+s}\right)^{-\frac{1}{n+1}} = 10^{m-2} \implies s = (2-m)(n+2)$$

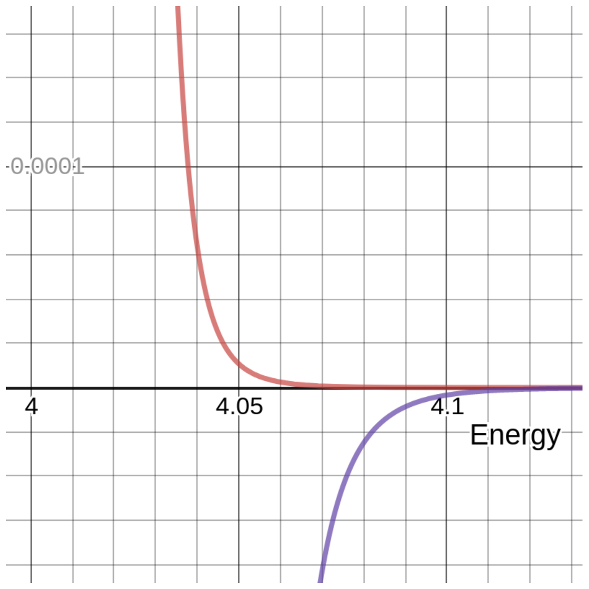

Figure 14: Eigenvalue Regularization

(a) Red : $f(E)$, Purple : $f'(E)$

Note how quickly the function flattens for the case $E_1 = 4, E_2 = 8$. The same behaviour is observed near $E = 8$ as well. The derivative grows faster than the function near the singularity.

5.2 E1 = 2.5, E0 = 3.25, E2 = 4

Firstly, it is helpful to observe the results without eigenvalue regularization. Consider case of $L = 4$. The following is the *eigenvalue* vs. *iterations* plot without controlling the eigenvalue range -

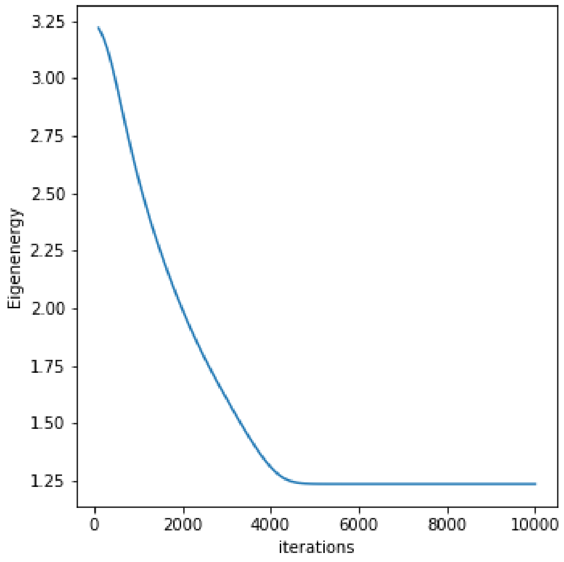

Figure 16: Eigenvalue Falling Through

The network falls down below $E_1 = 2.5$ and learns the eigenvalue for the $n = 2$ case. Now, with the eigenvalue regularization terms added, the plot changes as follows -

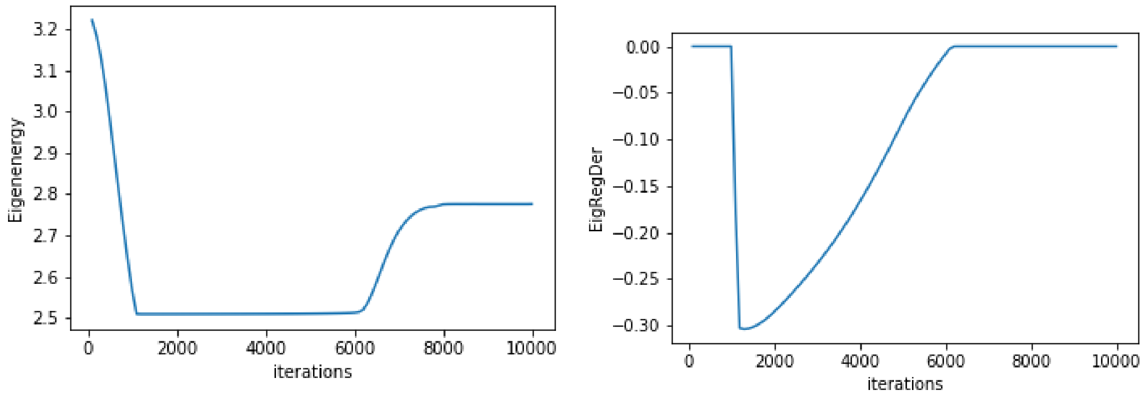

Figure 17: Plots of eigenvalue and its regularization derivative

It is clear how an increase in magnitude of the regularization derivative forces the eigenvalue to go up from 2.5 an stabilize at the energy level for $n = 3$.

5.3 E1 = 2, E0 = 2.2, E2 = 3

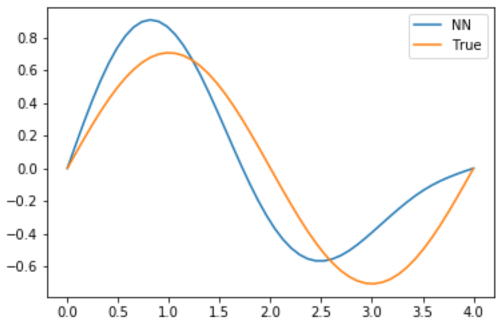

Figure 18: Failure of Eigenvalue Regularization

There is a significant difference between the network's output and the analytical answer. The eigenvalue learnt by network is 2.00536509 which does not correspond to any E_n for $L = 4$. For $n = 2$, $E_2 = 1.2337005$ and for $n = 3, E_3 = 2.775826$. It was hoped that the eigenvalue regularization would push the network back up to learn E_3 although it had a tendency to fall towards E_2. Although the probability is good 0.99845, the MSE is 0.14805912 which is unsatisfactory for this case of low $\frac{n}{L}$. Further insight can be obtained by looking at the curve of the eigenvalue regularization derivative.

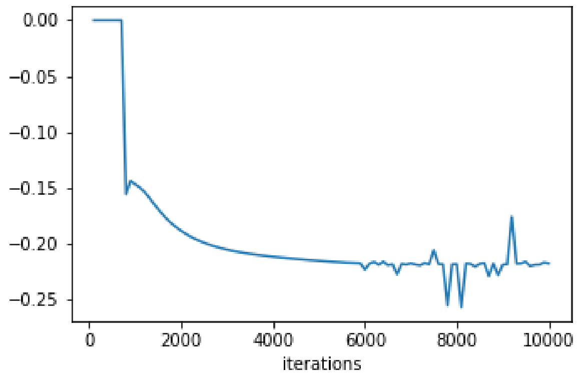

Figure 19: Failure of Eigenvalue Regularization

In the previous case, the regularization derivative flattens to zero over the final epochs. However, for the present case the derivative is still trying to push back the eigenvalue inside the range. This explains why the learnt eigenvalue didn't correspond to any E_n

6 Improvements

In this report, attempts have been made to generalise the neural network over multiple cases. However, it is noted that for particular cases the training is slower than others. This could be fixed by normalizing the differential equation for any value of L and then solving the system.

Currently, the entire training set is considered at once to optimize the loss, which slows down training over the epochs. This was deemed as necessary because the probability density integral needed to be calculated. However, it is possible to train the network in batches and perform the optimization. Currently *tanh* activation is used, but it suffers from the vanishing gradient. Other activations could be looked into.

It is possible to conceive of an architecture of two simultaneous neural networks - *one for the eigenfunction, and the other for the eigenvalue* - that is trained over a common loss function. It has been noticed that the eigenvalue changes in similar magnitudes irrespective of the scale of the system (L), which is clearly a problem. This could be overcome by having a separate neural network output the eigenvalue. In some sense, the nodes backing the eigenvalue output offers more control over its change, and it could dynamically take care of the scale of the problem. In the case of eigenvalue regularization, **`adam`** could be modified to externally increase eigenvalue derivative dynamically if it is noticed that it has spent many too many epochs near either of the singularity points.

Pertinent issues -

1. The choice of the energy parameter initialisation seems back-engineered and specific to the problem. Having a separate neural network control the eigenvalue output could fix the issue. Also it could well be extended to a generic Quantum Mechanics problem.
2. The decision of using ΔMSE_{100} for the exit condition could again be attributed to specificity. It seemed natural only *after* manually observing the training of multiple cases. It may not apply from all differential equations or potentials.
3. Due to the lack of control in the eigenvalue derivatives, it could be possible that it crosses any of singularities of the regularization terms, or lands exactly on it. There is no recovery from this situation in the present framework.
   (a) On a second thought, it could be possible to create infinite *"wells"* of regularization terms such as $\dfrac{1}{(E-E_1)^n} + \dfrac{1}{(E-E_2)^n} + \dfrac{1}{(E-E_3)^n} + \dfrac{1}{(E-E_4)^n}$

- This defeats the purpose of using regularization terms at all as the eigenvalue is free to assume any value. However, this could be a fruitful idea in the case of an unknown potential, whose eigenvalues are sought. It serve as a systematic way to divide the eigenvalue space and search for possible eigenfunction in them.

(b) Gradient clipping on the eigenvalue could be done near the singularity points to prevent the jump

(c) It is difficult to judge if the network has learnt the correct wavefunction or not. However, this could be solved by observing the eigenvalue regularization derivatives near the end of the training.

4. Particle in a Box is further simplified by the fact that the potential is infinity outside the box. We know *a priori* that the wavefunction vanishes in such regions. However, for the generic potential, the square integrability of the wavefunction has to be checked for $x \to \pm\infty$. Using any numerical integration technique for the entire position space could slow down the training immensely.