24/02/2025

Meeting Summary

I will explain how I decided the points for each loss and how each is loss is calculated. Note that the code is uploaded in the box with the name “tdse_2022shah.ipynb”.

As you may know, here we are considering three main losses. Physics loss (L_{PDE}), Initial condition loss (L_{IC}) and Boundary condition loss (L_{BC}). Inorder to do this, we have to use random uniform points inside our domain. Quoting from the paper “*We used a batch size of 3140 points for the interior, 200 points for the boundary conditions and 314 points for initial conditions.*”.

Let’s start with interior collocation points which will be used to calculate the physics loss:

```
#pde loss
with tf.GradientTape(persistent=True) as tape:
    tape.watch([x_collocation_tf, t_collocation_tf])

    psi_real, psi_img = model((x_collocation_tf, t_collocation_tf))
    psi = tf.complex(psi_real, psi_img)
    psi = tf.cast(psi, tf.complex128)

    d_psi_dx = tape.gradient(psi, x_collocation_tf)
    d_psi_dt = tape.gradient(psi, t_collocation_tf)

d2_psi_dx2 = tape.gradient(d_psi_dx, x_collocation_tf)

del tape

i = tf.complex(0.0, 1.0)
i = tf.cast(i, dtype=tf.complex128)
d_psi_dt = tf.cast(d_psi_dt, tf.complex128)
d2_psi_dx2 = tf.cast(d2_psi_dx2, tf.complex128)

residual = i * d_psi_dt + 0.5 * d2_psi_dx2 - tf.cast(0.5 * (x_collocation_tf ** 2), tf.complex128) * psi
physics_loss = tf.reduce_mean(tf.square(tf.abs(residual)))
```

(variable) x_collocation_tf: Any

This is the block of code in which I calculated the physics loss. Here, *x_collocation_tf* will be 3140 random uniform points within the range (-π, π). Similarly, *t_collocation_tf* will be the 3140 random uniform points within the range (0, 2π). We will then give this data as inputs to the model to get 3140 wavefunctions, for each pair of x and t.

For example, if x_collocation_tf = [-3, -2, -1, …] and t_collocation_tf = [0, 1, 2, …], then the model will take input as model(-3, 0), model(-2, 1), model(-1, 2), … etc, and it will return the corresponding wavefunctions which we will use to compute residual. Note that, we will be having 3140 residuals, one for each wavefunction we tried to predict using each pair of 3140 x and t values. Then we will take the means squared error (MSE) of this residuals and that is the value of *physics_loss* for that iteration (epoch).

Similarly, we will calculate the initial condition loss.

```
#initial condition loss
psi_real_initial, psi_img_initial = model((x_initial_tf, t_initial_tf))
psi_initial = tf.complex(psi_real_initial, psi_img_initial)
psi_initial = tf.cast(psi_initial, tf.complex128)

psi_0 = ((1 / np.pi) ** 0.25) * tf.exp(-0.5 * (x_initial_tf ** 2))
psi_1 = psi_0 * np.sqrt(2) * x_initial_tf
psi_initial_actual = (psi_0 + psi_1) / np.sqrt(2)
psi_initial_actual = tf.cast(psi_initial_actual, tf.complex128)

initial_condition_loss = tf.reduce_mean(tf.square(tf.abs(psi_initial - psi_initial_actual)))
```

As per the paper, we need to use 314 random uniform points to calculate the initial condition loss. Here, *x_initial_tf* will be 314 random uniform points within the range (- π, π). *t_initial_tf* will an array of 0s with length 314 (since we a doing the initial condition were t=0). Then, we will calculate the wavefunction for each pair of x and t, like we did for the physics loss calculation.

Now we will have *psi_initial*, which will be an array of 314 wavefunctions, one for each pair of x and t (since all of the t will be 0, we can say this will be initial wavefunction predicted by the model for each x value). Now we will calculate the analytical initial condition for each x in *x_initial_tf*, which is the *psi_initial_actual*. Then we will take the MSE of psi_initial and psi_initial_actual which will be the initial_condition_loss for the iteration (epoch).

PLEASE NOTE THAT THE 314 POINTS I MENTIONED HERE IS DIFFERENT FROM AND NOT INCLUDED IN THE 3140 POINTS WHICH I USED TO CALCULATE THE PHYSICS LOSS.

Finally, we need to calculate the boundary condition loss.

```
#boundary condition loss
psi_real_boundary, psi_img_boundary = model((x_boundary_tf, t_boundary_tf))
psi_boundary = tf.complex(psi_real_boundary, psi_img_boundary)
psi_boundary = tf.cast(psi_boundary, tf.complex128)
boundary_condition_loss = tf.reduce_mean(tf.square(tf.abs(psi_boundary)))
```

Here, the paper used 200 points to calculate the boundary condition loss. *x_boundary_tf* will be an array of size 200, in which first 100 values will be -π and next 100 values will be π (spatial boundaries). Now, *t_boundary_tf* will be an array of 200 random uniform values between the range (0, 2π). Then, like we did in the last two losses, the model will predict 200 wavefunctions, for each pair of x and t and we store it in the variable *psi_boundary*. Since we are using Dirichlet boundary conditions, the actual wavefunction should be 0 for all 200 points. Hence, we will take the MSE of the *psi_boundary* as the *boundary_condition_loss* since our main goal is to make *psi_boundary* close to zero.

Once again, we select the 200 points to calculate the boundary_condition_loss separately and this points are not part of the points we took to calculate the physics loss and initial condition loss.

After we calculated the three individual losses, we then calculate our final loss (total loss) as the sum of the individual losses. That is, $L = L_{PDE} + L_{IC} + L_{BC}$

This will the loss of a single iteration, which will be used to update the weights and biases of the model. This cycle will repeat for a number of times (eg. 150000) till the rate of change of L becomes close to zero.