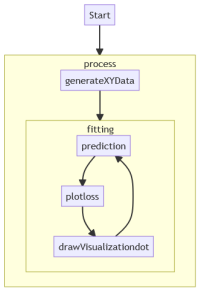
物聯網實務第十二周作業

電機四乙10828241 陳大荃

December 5, 2022

Exercise 11-1 Visualize TF gradient descent.

Full detail in GitHub

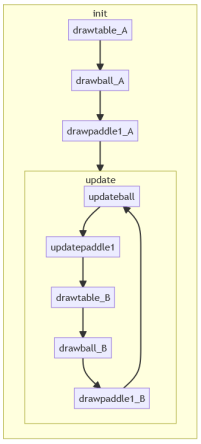


Detail

1. process()
    1. generate initial linear regression coefficient.
    2. call generateXYData() to retreive training data sets.
    3. feed training datasets to fitting()
2. generateXYData()
    1. generate x data in the range between 0 and 1.
    2. generate y data based on corresponding x value and a random component.
    3. return two datasets: x, y.
3. fitting()
    1. set the method to construct neural network as sequential instead of functional. src
    2. add a dense layer
        1. tf.layers.dense.units: neuron count
        2. tf.layers.dense.inputShape: input node count
    3. set training optimizer as tf.SGDOptimizer which utilizes stochastic gradient descent(extreme large dataset).
    4. configure model settings for training.
    5. add bias node for input layer.
    6. convert datasets to tensor sets.
    7. start training.
        1. tf.keras.model.fit.batchSize: the number of samples to work through before updating the internal model parameters.
        2. tf.keras.model.fit.epochs: the number times that the learning algorithm will work through the entire training dataset.
        3. tf.keras.model.fit.callbacks: access point of the model during training. In this case, it does the following stuff.
            1. tf.keras.callbacks.Callback.onEpochEnd: called at the end of Epoch, and perform the following stuff.
                1. print out epoch count and log.
                2. call Prediction().
                    1. prepare x data (input) for prediction (add bias node).
                    2. turn prepared x data into tensor.
                    3. tf.keras.model.predict: use input tensor to predict with current iteration of NN.
                    4. tf.Tensor.dataSync: retrieve data from tensor.
                    5. tf.keras.model.dispose: delete model.
                    6. tf.Tensor.dispose: delete predict result from memory.
                3. call plotData2().
                    1. restructure input array.
                    2. create vegaEmbed data structure containing restructured array as data and plot it on webpage.
                4. call plotloss().
                    1. restructure input array.
                    2. create vegaEmbed data structure containing restructured array as data and plot it on webpage.
                5. extract NN node current weight and log.
                6. append newly retrieve data.
                7. call drawVisualizationdot()
                    1. create vis.Graph3d data structure.
                    2. draw graph.
            2. jump back to 1, and loop until the training is finished.

Exercise 11-2 Build the fundation of the pong game.

Full detail in GitHub

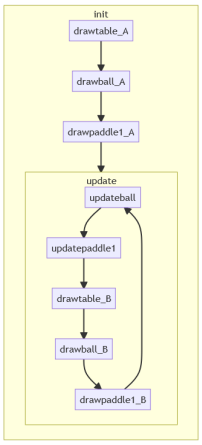


Detail

1. call init()
    1. call drawtable()
        1. draw the table background.
        2. define the table border.
    2. call drawball()
        1. context.beginPath: initialize current subpath (drawing).
        2. context.arc: draw a circular object.
        3. fill the defined circle, which is the ball.
    3. call drawpaddle1()
        1. define a rectangle shaped object as the ball-reflecting paddle.
        2. fill the defined shape
    4. loop update() with a designated interval as 60 times per second.
        1. call updateball()
            1. define the top, bottom, left, right coordinate of the ball based on its center coordinate.
            2. if the ball (most left/right) is somehow outside of the defined left/right wall, invert its moving direction along the x axis. (simulate bouncing)
            3. if the ball (most top/bottom) is somehow outside of the defined upper wall/paddle, invert its moving direction along the y axis. (simulate bouncing)
            4. if the ball (center) is below the defined table (failed to reflect the ball by defined paddle), reset the ball coordinate. if not, update the moving speed of the ball.
        2. call udpatepaddle1()
            1. if left arrow key is pressed, move paddle left.
            2. if right arrow key is pressed, move paddle right.
            3. if not left or right arrow key is pressed, stop paddle from moving.
            4. if paddle moves outside of the left/right wall, limit their movement.
        3. call drawtable()
            1. Re-draw the table.
        4. call drawball()
            1. Re-draw the ball.
        5. call drawpaddle1()
            1. Re-draw the paddle.
2. record any key pressed movement.
    1. store the key in array.
    2. print out what key is pressed currently.
3. record any key lifted movement.
    1. remove the key in the key storage array.

Exercise 11-3 Gather player's control data for later use.

Full detail in GitHub

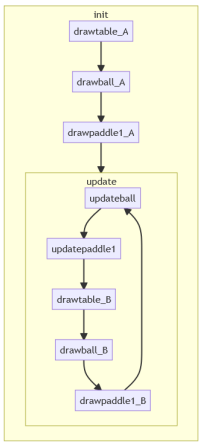


Detail

1. call init()
    1. call drawtable()
        1. draw the table background.
        2. define the table border.
    2. call drawball()
        1. context.beginPath: initialize current subpath (drawing).
        2. context.arc: draw a circular object.
        3. fill the defined circle, which is the ball.
    3. call drawpaddle1()
        1. define a rectangle-shaped object as the ball-reflecting paddle.
        2. fill the defined shape
    4. loop update() with a designated interval 30 times per second.
        1. call updateball()
            1. define the top, bottom, left, right coordinate of the ball based on its center coordinate.
            2. if the ball (most left/right) is somehow outside of the defined left/right wall, invert its moving direction along the x-axis. (simulate bouncing)
            3. if the ball (most top/bottom) is somehow outside of the defined upper wall/paddle, invert its moving direction along the y-axis. (simulate bouncing)
                1. if it's hitting the top wall, discard the player decision data.
                2. if it's hitting the paddle, increase player strike time.
            4. if the ball (center) is below the defined table (failed to reflect the ball by defined paddle), reset the ball coordinate, discard the player decision data. if not, update the moving speed of the ball.
            5. store paddle and ball coordinate.
            6. store player decision as either 0(left), 1(none), 2(right).
            7. store combine current and previous paddle and ball coordinate in array.
            8. action when player hit the ball with paddle.
                1. if it's the first time, clear all the previous data in memory.
                2. if it's not the first time, store all the previously stored data (5,6,7) in massive data array.
                3. if the number of data stored in massive data array exceed designated amount (10 in this case), create x (input: ball_x, ball_y, paddle_x, old_ball_x, old_ball_y, old_paddle_x) and y (output: left, none, right) data.
                4. convert the x and y array into JSON format.
                5. pop the download window and let user select local directory to store the JSON file.
            9. if the data is flagged to be cleaned, clean it.
        2. call udpatepaddle1()
            1. if left arrow key is pressed, move paddle left.
            2. if right arrow key is pressed, move paddle right.
            3. if not left or right arrow key is pressed, stop paddle from moving.
            4. if paddle moves outside of the left/right wall, limit their movement.
        3. call drawtable()
            1. Re-draw the table.
        4. call drawball()
            1. Re-draw the ball.
        5. call drawpaddle1()
            1. Re-draw the paddle.
2. record any key-pressed movement.
    1. store the key in array.
    2. print out what key is pressed currently.
3. record any key-lifted movement.
    1. remove the key in the key storage array.

Exercise 11-4 Gather more player's control data for later use.

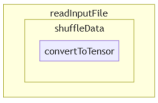
Full detail in GitHub



Gather more data then exercise 11-3.

Exercise 11-5 Use gathered data to train NN model.

Full detail in GitHub

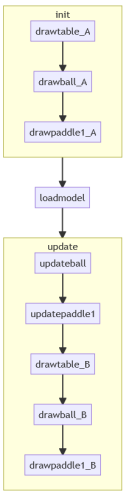


Detail

1. Call readInputFile() if the user clicked "browse" to upload the model stored by both "ex11_3.html" and "ex11_4.html".
    1. From the uploaded model file, read the content from it.
    2. Turn the content into a JSON object.
    3. Call shuffleData()
        1. Separate different subobjects into x and y and recombine them.
        2. tf.util.shuffle: shuffle the x and y datas.
        3. Call convertToTensor()
            1. Separate x and y sets.
            2. Get length of the sets.
            3. Turn x and y sets into separate 2D tensor arrays.
            4. Create a model in the shape of 66464*3 (without bias, include input and output).
            5. Set model optimizer as "adam" and set learning rate, loss, and output number during training.
            6. Print out the structure of the created model.
            7. Fit (train) the constructured model with designated batchSize, and epochs settings.
            8. Save the model in browser memory.
            9. Release memory storaging x and y datasets.

Exercise 11-6 Use trained model to interact with player.

Full detail in GitHub



Detail

1. call init()
    1. call drawtable(), to draw the pong table.
    2. call drawball(), to draw the pong.
    3. call drawpaddle1(), to draw the ball-reflecting paddle.
2. call loadmodel()
    1. load trained model from webpage memory storage.
    2. print out the structure of model.
    3. call update() 60 times per second.
        1. call updateball()
            1. determine whether the ball hit both sides of the table.
                1. if hitting the right wall, turn x direction left.
                2. if hitting the left wall, turn x direction right.
            2. determine whether the ball hit the upper wall or the paddle.
                1. if hitting the upper wall, turn y direction down.
                2. if hitting the paddle, turn y direction up.
            3. determine whether the ball exceed the bottom of the table.
                1. if exceed, reset ball position.
                2. if not exceed, update ball location.
        2. call updatepaddle1()
            1. store current data: paddle_x_pos, ball_x_pos, ball_y_pos.
            2. combine current data and previous data.
            3. turn these combined data into tensors.
            4. send the tensors to loaded model and predict with it.
            5. extract the prediction and user the result to move the paddle.
            6. determine whether the paddle positionn exceed left and right walls. if exceed, limit it's position between the two walls.
            7. delete tensors and prediction resulst.
        3. call drawtable(), to draw the pong table.
        4. call drawball(), to draw the pong.
        5. call drawpaddle1(), to draw the ball-reflecting paddle.

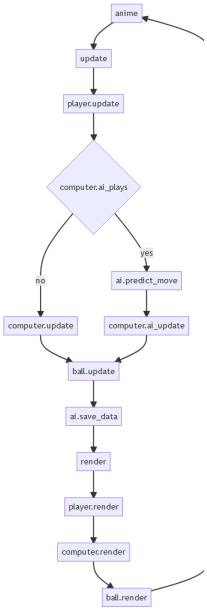
Exercise 11-7 Gather player data while playing and use them to train NN realtime.

Full detail in GitHub

Detail (ex11_7_pong_game.js)

(this file is launched after loading all tensorflow functions)

1. Initialize NN model.
2. Add canvas to html document.
3. call step() to loop for 60 times per second by animate().
    1. call update().
        1. call player.update()/Player.prototype.update()
            1. if left arrow key is pressed, move player paddle left.
            2. if right arrow key is pressed, move player paddle right.
            3. if other keys is pressed, don't move player paddle.
        2. determine whether computer.ai_plays is true. (whether ai is running)
            1. if it's true
                1. perform prediction with the trained model.
                2. extract the decision of the model.
                3. move computer controlled paddle.
            2. if not
                1. calculate the x-axis distance between the ball and the computer controlled paddle.
                2. move the computer controlled paddle based on the calculated distance.
        3. call ball.update()
            1. update ball coordinate based on the speed of the ball.
            2. update ball border by the new ball coordinate.
            3. revert the ball x-axis moving direction if the outline of ball touches left and right walls.
            4. if the ball goes outside of the y-axis limit (top/bottom table)
                1. reset ball speed and coordinate.
                2. call ai.new_turn()
                    1. clear previous data.
                    2. update turn count.
                    3. if it's not the first turn
                        1. call ai.train()
                            1. if no new data is store, return this function
                            2. turn all the newly gathered data into tensors. (both x, y)
                            3. train the model with the updated tensors.
                        2. call ai.reset()
                            1. reset all parameters.
            5. determine which section the ball is resting
                1. in player's section
                    1. if the ball hit player's paddle, invert the ball's y direction.
                2. in computer's section
                    1. if the ball hit computer's paddle, inver the ball's y direction.



4. call ai.save_data()
    1. if the NN model is not initialized yet, return this function.
    2. if no previous data is stored, store and return.
    3. based on whether the ball traveled to different side of the table
        1. if yes, keep reverted x(input) data
        2. if no, keep x(input) data
    4. store the data to memory for future training.
2. call render()
    1. draw the table.
    2. call player.render()
        1. draw player's paddle.
    3. call computer.render()
        1. draw computer's paddle.
    4. call ball.render()
        1. draw the ball.