Sample High Low close Date GBPUSD

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Date** |  | **Open** | **High** | **Low** | **Close** | **Volume** |
|  |  |  |  |  |  |  |
| 1993.05.12 | 00:00 | 1.537 | 1.5445 | 1.529 | 1.5338 | 2781 |
| 1993.05.13 | 00:00 | 1.5328 | 1.536 | 1.518 | 1.5225 | 2571 |
| 1993.05.14 | 00:00 | 1.5228 | 1.5415 | 1.52 | 1.5387 | 2711 |
| 1993.05.17 | 00:00 | 1.5365 | 1.546 | 1.5309 | 1.5355 | 2921 |
| 1993.05.18 | 00:00 | 1.535 | 1.538 | 1.5237 | 1.5365 | 2711 |
| 1993.05.19 | 00:00 | 1.535 | 1.5482 | 1.5328 | 1.5432 | 2261 |
| 1993.05.20 | 00:00 | 1.5425 | 1.5603 | 1.5383 | 1.5565 | 3001 |
| 1993.05.21 | 00:00 | 1.5548 | 1.5592 | 1.539 | 1.5425 | 2811 |
| 1993.05.24 | 00:00 | 1.54 | 1.545 | 1.5289 | 1.5365 | 2871 |
| 1993.05.25 | 00:00 | 1.5385 | 1.547 | 1.5345 | 1.542 | 2151 |
| 1993.05.26 | 00:00 | 1.542 | 1.5505 | 1.541 | 1.5472 | 1381 |
| 1993.05.27 | 00:00 | 1.5453 | 1.565 | 1.5425 | 1.562 | 2831 |
| 1993.05.28 | 00:00 | 1.561 | 1.568 | 1.5535 | 1.5607 | 2871 |
| 1993.05.31 | 00:00 | 1.5605 | 1.563 | 1.556 | 1.561 | 1351 |
| 1993.06.01 | 00:00 | 1.5613 | 1.567 | 1.546 | 1.5555 | 3621 |
| 1993.06.02 | 00:00 | 1.5544 | 1.556 | 1.5364 | 1.54 | 2481 |
| 1993.06.03 | 00:00 | 1.5403 | 1.5501 | 1.537 | 1.5501 | 1641 |
| 1993.06.04 | 00:00 | 1.549 | 1.5505 | 1.506 | 1.5085 | 4851 |
| 1993.06.07 | 00:00 | 1.5063 | 1.5268 | 1.506 | 1.5268 | 2111 |
| 1993.06.08 | 00:00 | 1.5253 | 1.5265 | 1.5135 | 1.5205 | 2121 |
| 1993.06.09 | 00:00 | 1.5195 | 1.527 | 1.5069 | 1.5152 | 3591 |
| 1993.06.10 | 00:00 | 1.5137 | 1.5325 | 1.507 | 1.5305 | 3421 |
| 1993.06.11 | 00:00 | 1.5293 | 1.5405 | 1.5185 | 1.5215 | 3621 |
| 1993.06.14 | 00:00 | 1.519 | 1.533 | 1.519 | 1.5285 | 1851 |
| 1993.06.15 | 00:00 | 1.5275 | 1.5355 | 1.514 | 1.5165 | 3201 |

**Using code from ConvetJs Sample**

**I have highlighted in Yellow the possible variation we can use for project this is just for illustration, the original code untouched as used reference**

[**https://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html**](https://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html)

var num\_inputs = 27; // 9 eyes, each sees 3 numbers (wall, green, red thing proximity)

**var num\_inputs = 210\*160 RGB; // Pixel Data**

var num\_actions = 5; // 5 possible angles agent can turn

**var num\_actions = 3; // Buy Open (TIME), Exit Close(TIME), Sell Open (TIME), Exit Close), Do Nothing (TIME)**

var temporal\_window = 1; // amount of temporal memory. 0 = agent lives in-the-moment :)

**var temporal\_window = 1; // amount of temporal memory. 0 = agent lives in-the-moment :) Not sure how this would integrate with out systems**

var network\_size = num\_inputs\*temporal\_window + num\_actions\*temporal\_window + num\_inputs;

**var network\_size = num\_inputs\*temporal\_window + num\_actions\*temporal\_window + num\_inputs;**

// the value function network computes a value of taking any of the possible actions

// given an input state. Here we specify one explicitly the hard way

// but user could also equivalently instead use opt.hidden\_layer\_sizes = [20,20]

// to just insert simple relu hidden layers.

var layer\_defs = [];

layer\_defs.push({type:'input', out\_sx:1, out\_sy:1, out\_depth:network\_size});

layer\_defs.push({type:'fc', num\_neurons: 50, activation:'relu'});

layer\_defs.push({type:'fc', num\_neurons: 50, activation:'relu'});

layer\_defs.push({type:'regression', num\_neurons:num\_actions});

// options for the Temporal Difference learner that trains the above net

// by backpropping the temporal difference learning rule.

var tdtrainer\_options = {learning\_rate:0.001, momentum:0.0, batch\_size:64, l2\_decay:0.01};

var opt = {};

opt.temporal\_window = temporal\_window;

opt.experience\_size = 30000;

opt.start\_learn\_threshold = 1000;

opt.gamma = 0.7;

opt.learning\_steps\_total = 200000;

opt.learning\_steps\_burnin = 3000;

opt.epsilon\_min = 0.05;

opt.epsilon\_test\_time = 0.05;

opt.layer\_defs = layer\_defs;

opt.tdtrainer\_options = tdtrainer\_options;

var brain = new deepqlearn.Brain(num\_inputs, num\_actions, opt); // woohoo

It's very simple to use deeqlearn.Brain: Initialize your network:

var brain = new deepqlearn.Brain(num\_inputs, num\_actions);

And to train it proceed in loops as follows:

var action = brain.forward(array\_with\_num\_inputs\_numbers);

// action is a number in [0, num\_actions) telling index of the action the agent chooses

// here, apply the action on environment and observe some reward. Finally, communicate it:

brain.backward(reward); // <-- learning magic happens here

**brain.backward(reward); // <-- learning magic happens here maximum pips overtime**

That's it! Let the agent learn over time (it will take opt.learning\_steps\_total), and it will only get better and better at accumulating reward as it learns. Note that the agent will still take random actions with probability opt.epsilon\_min even once it's fully trained. To completely disable this randomness, or change it, you can disable the learning and set epsilon\_test\_time to 0:

brain.epsilon\_test\_time = 0.0; // don't make any random choices, ever

brain.learning = false;

var action = brain.forward(array\_with\_num\_inputs\_numbers); // get optimal action from learned policy

**Notes,**

After reading white paper to further enhance the probability of success the following observation where made watching algorithm play computer games

1) Forex CSV chart data that is upload to website needs to be converted to pixel data, we can then user higher resolution data, to act as an input example 1 minute charts right up to daily.

2) Change the buy open and exit on close to a time based option, example –

**Start time = 00:00:01 and Trade duration = 24 Hours** (trade open at 00:01 and closed 24 hours later)

If the above can be implement, I believe there is a number of advantages

1. A complete array of indicators and be added if and when required
2. Trade can be created for anytime frame instantly
3. It would work more closely to the original model that is presented in white paper

Challenge is now to convert input data into pixels, and use the appropriate co-ordinates to feed network, as input