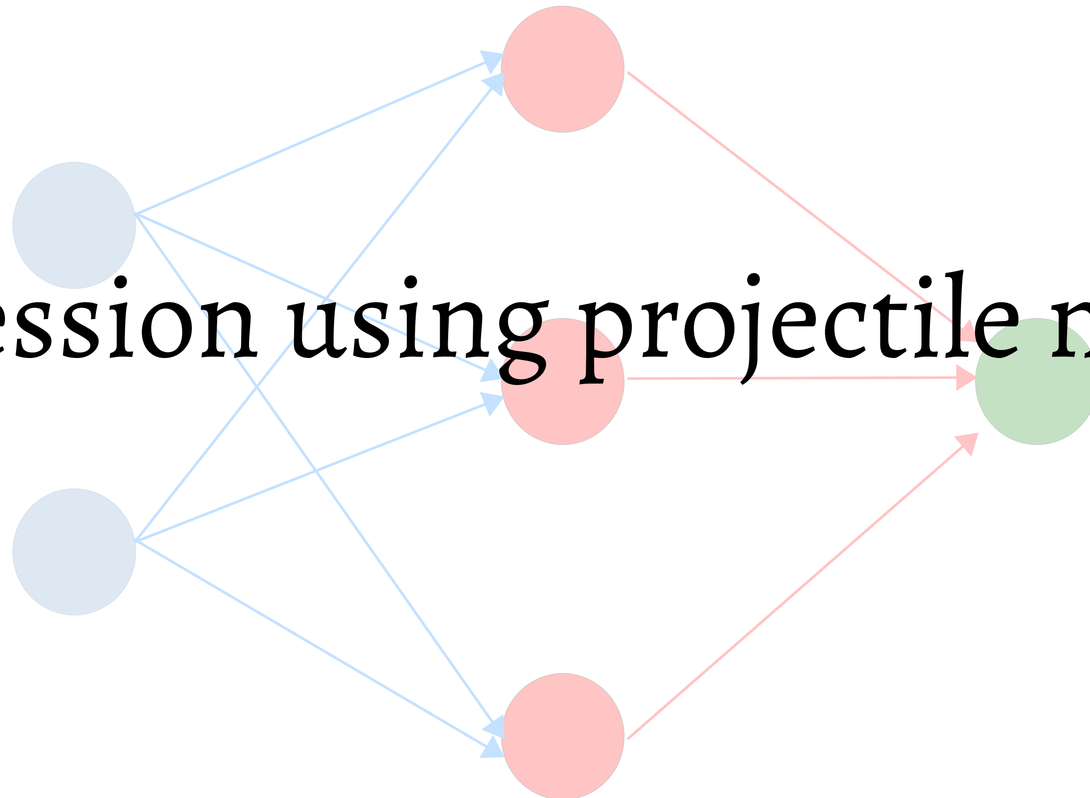
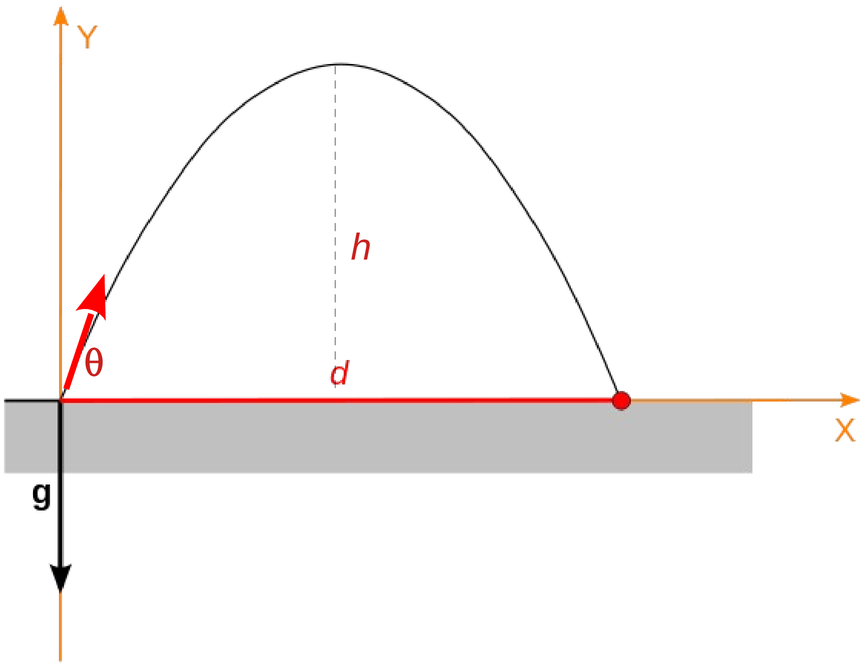
PH6232: Machine Learning for Physics applications

Regression using projectile motion



Feb 10th, 2023

Setup of task



Distance = $v^2 \sin(2\theta) / g$

Height = $v^2 \sin^2(\theta) / (2g)$

Time of Flight = $2 v \sin(\theta) / g$

Two inputs, v and angle (θ) determine the other three parameters

Our task

```
(base) SoL~/Sourabh/Work/Documents/Courses/PH6232_ML/projectile>more input.txt
1.9061 8.90143 16.1042 12.5943 47.868
1.50155 5.52391 13.9021 11.826 38.4736
1.67037 6.83587 14.9818 12.1423 42.3823
1.8526 8.4087 27.0152 17.177 31.9029
1.62643 6.48093 11.9861 10.8547 47.2398
2.1155 10.9646 31.7486 18.2395 34.6335
2.92783 21.0019 39.6969 19.7396 46.6174
2.38901 13.9831 33.9819 18.4218 39.4534
1.32226 4.2835 14.3245 12.623 30.8823
2.11461 10.9553 30.4389 17.736 35.7472
1.63993 6.58899 17.4809 13.349 37.0109
2.83727 19.7227 34.8304 18.5468 48.5555
2.26943 12.6183 25.9501 15.9502 44.2014
1.75516 7.54745 25.7986 17.0299 30.3322
1.49742 5.49359 12.0236 10.8771 42.4211
```

projectile_input_200.txt : (get it from the git folder)

It has 5 columns: tof, height, dist, v, angleDeg

It has 200 entries (*i.e. 200 examples*)

Our task: Based on the tof, height, dist

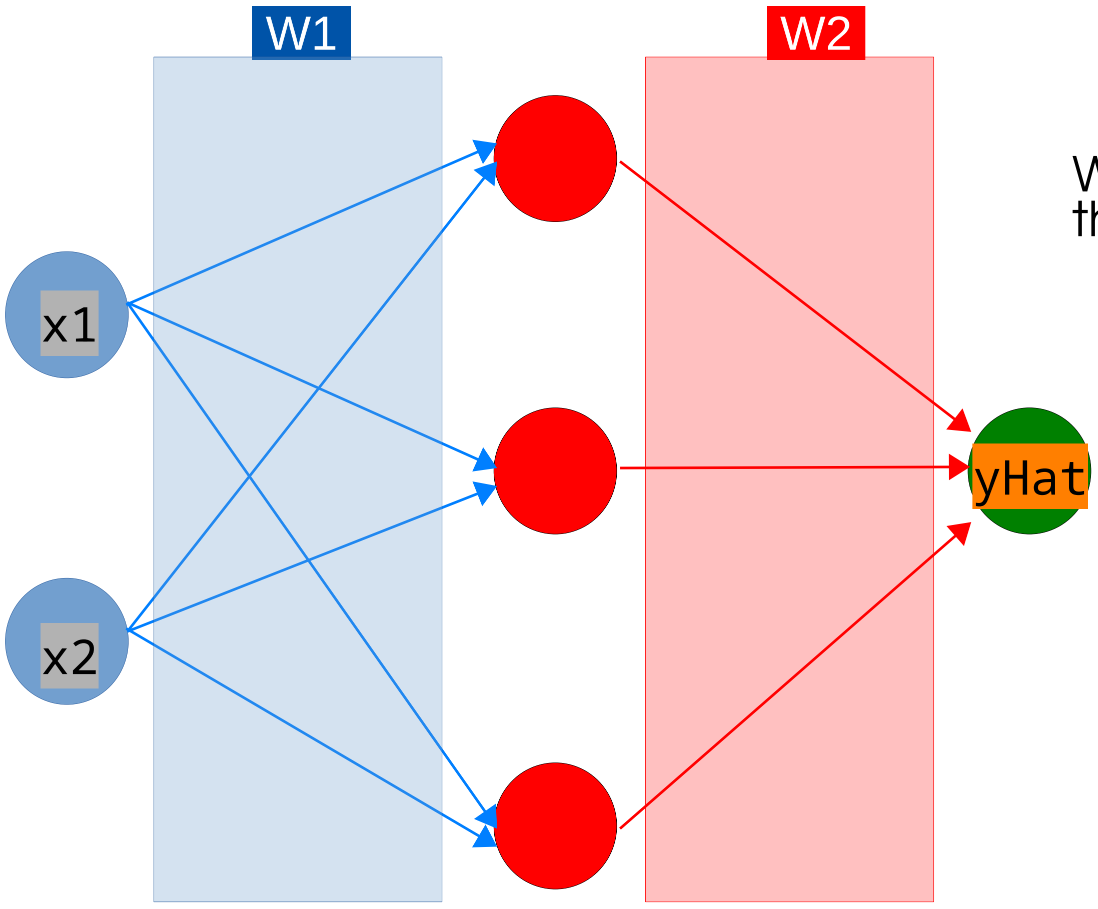
can we predict the initial velocity?

Workflow

1. Read in the input file in a dataframe (read in first four columns, ignore `angleDeg`)
2. Visualize the inputs (make plots). This is important.
3. Split the dataframe into input vector (X) and targeted output (y)
   Here X will have `tof, height, distance` and y will have `velocity`
4. Define a neural network.
   Choose architecture, choose loss function, etc.
5. Train for a certain number of epochs.
6. Save the model. (Visualize the training....)
7. Read a test file (`projectile_input_test_10.txt`)
   (A test has examples that are not part of the input file)
   Split into `inputs, true_output`.
   Based on the `inputs`, make `predicted_output`
   Compare `true_output` and `predicted_output`

8. Now play around till the network starts to perform better.

Gradient descent



What combination of weights gives the best predictions?

Cost function

What combination of weights gives the best predictions?

We could change W1 and W2 and recalculate yHat.. how do we know if its better?

Define a cost function

Lower the cost, better the prediction..

We need to find the combination of weights that minimizes the cost.

$$J = \Sigma \frac{1}{2}(y - \hat{y})^2$$

Gradient descent

We have cost as a function of the weights.

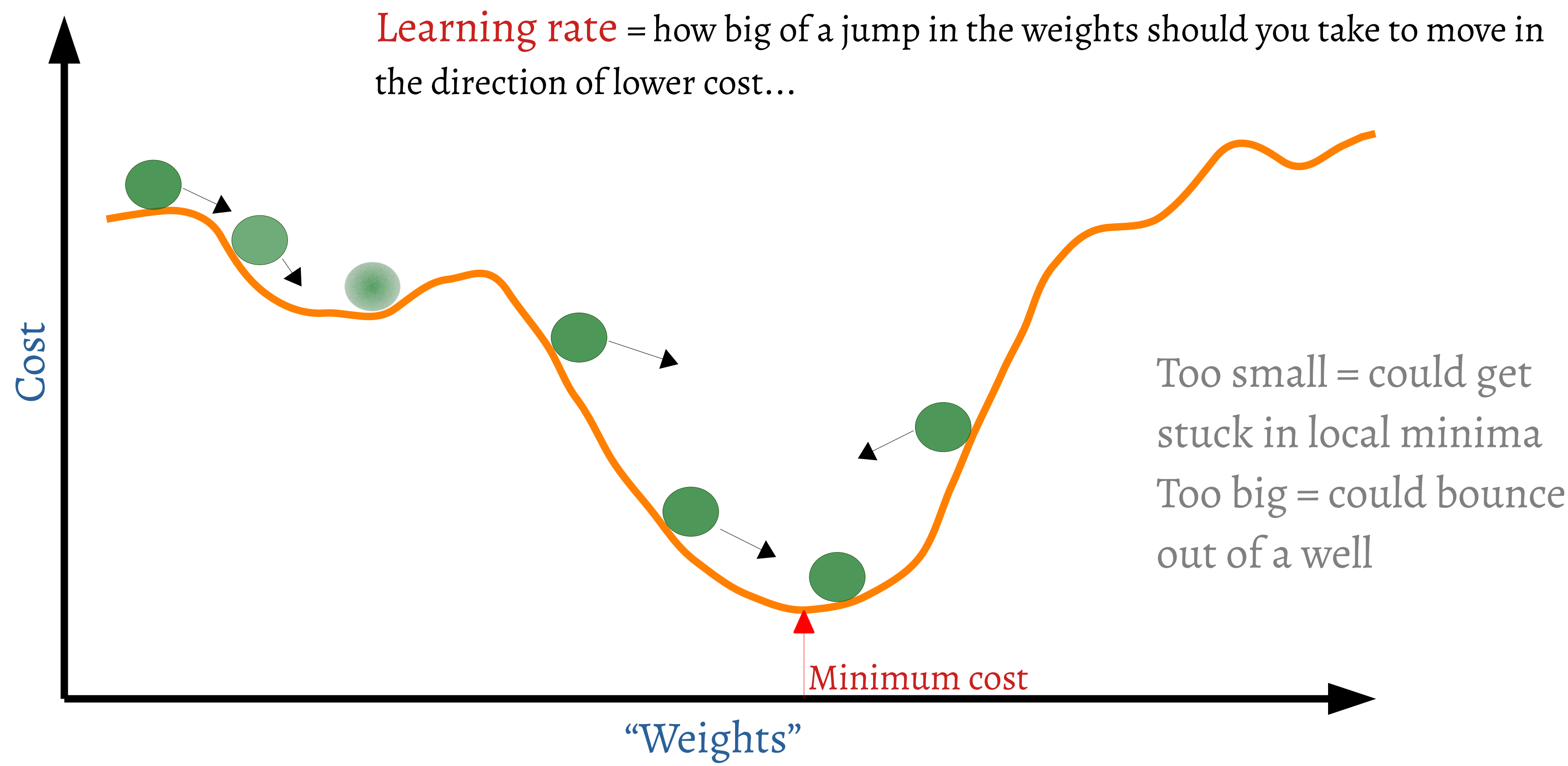
We calculate the gradient of cost w.r.t. weights, identifying the direction in which J is reducing.

$$\frac{\partial J}{\partial W_1}, \frac{\partial J}{\partial W_2}$$

We change the weights by some amount (Rate $\times$ Gradient), and recalculate cost.

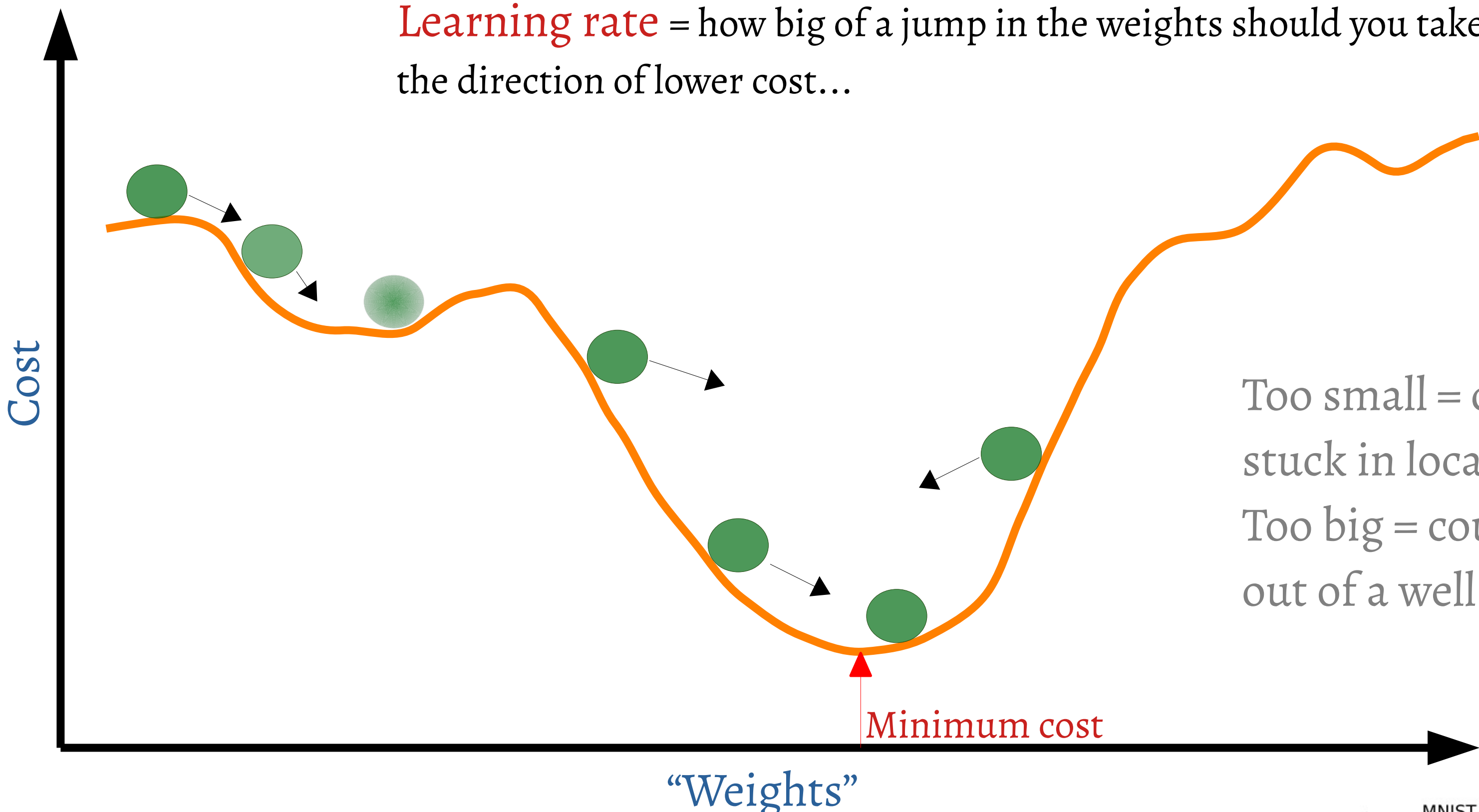
And thus iterate until we are satisfied with the weights…

Gradient descent



Gradient descent

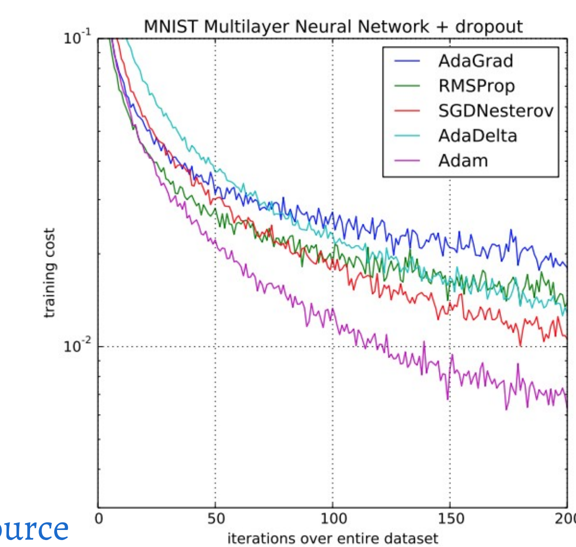
Learning rate = how big of a jump in the weights should you take to move in the direction of lower cost…

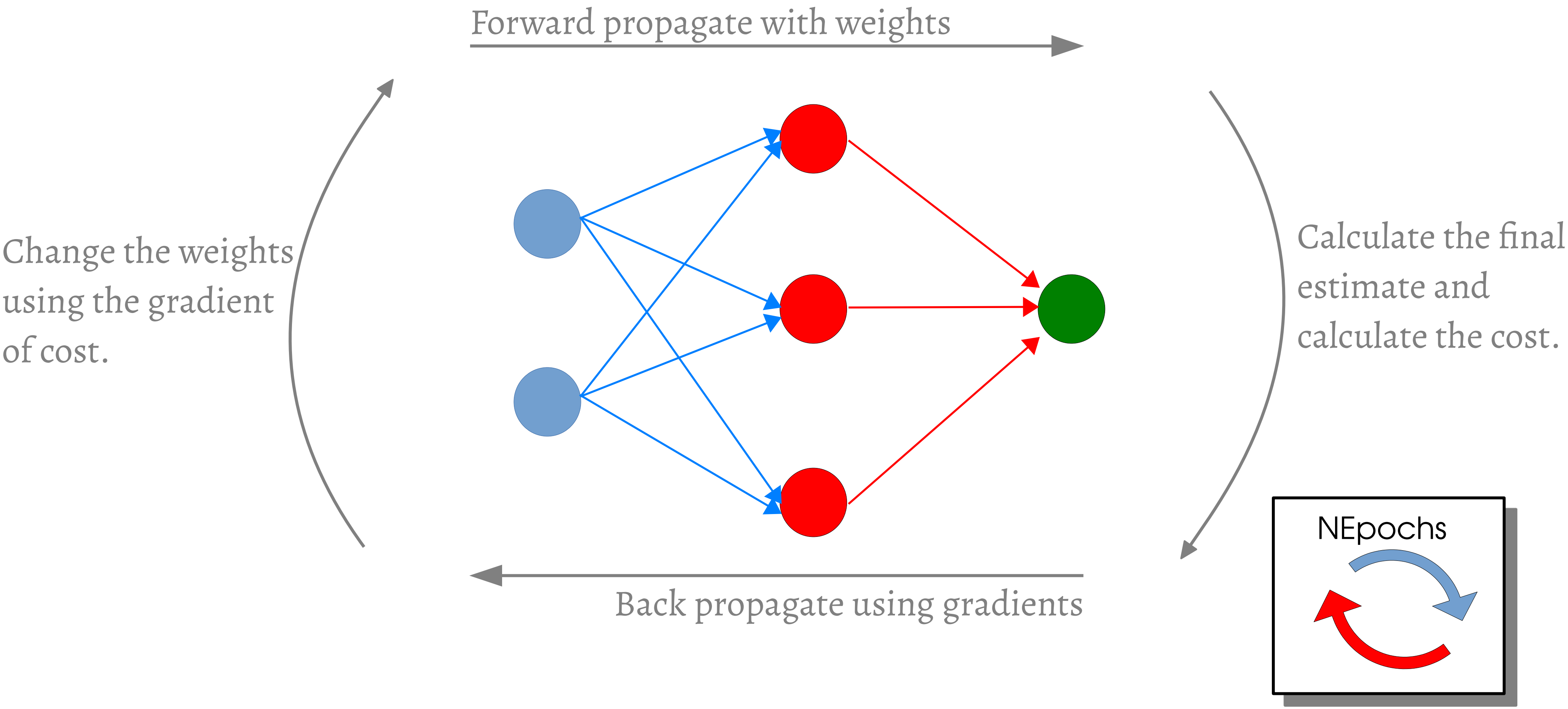


Too small = could get stuck in local minima
Too big = could bounce out of a well

In practice, we use a pre-defined learning rate optimizer called 'adam'
Adam (adaptive moment estimation)
[https://arxiv.org/abs/1412.6980]

source

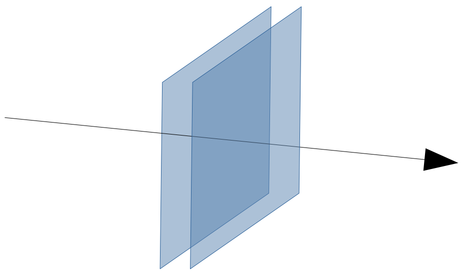




Batch size : the number of examples to see before calculating costs and updating weights

Epoch : when all examples are seen once by the network.

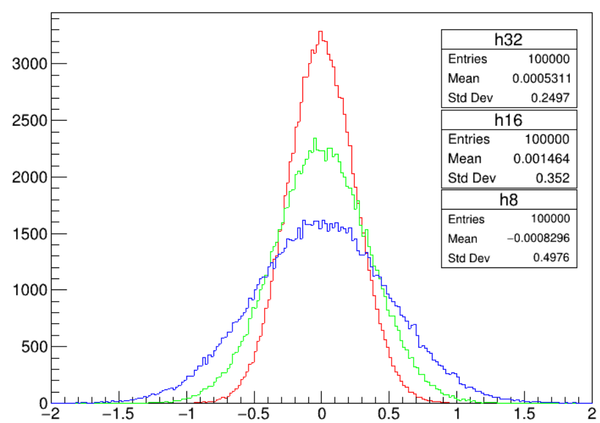
Starting network for today



Two hidden layers, with 4 and 2 neurons respectively.

Activation function for each neuron: `ReLU`

The weights are initialized as `he_normal`



Normal distribution with (0,σ)

$\sigma = (2/\text{inputNeurons})^{1/2}$

Loss function: `'MeanSquaredError'`

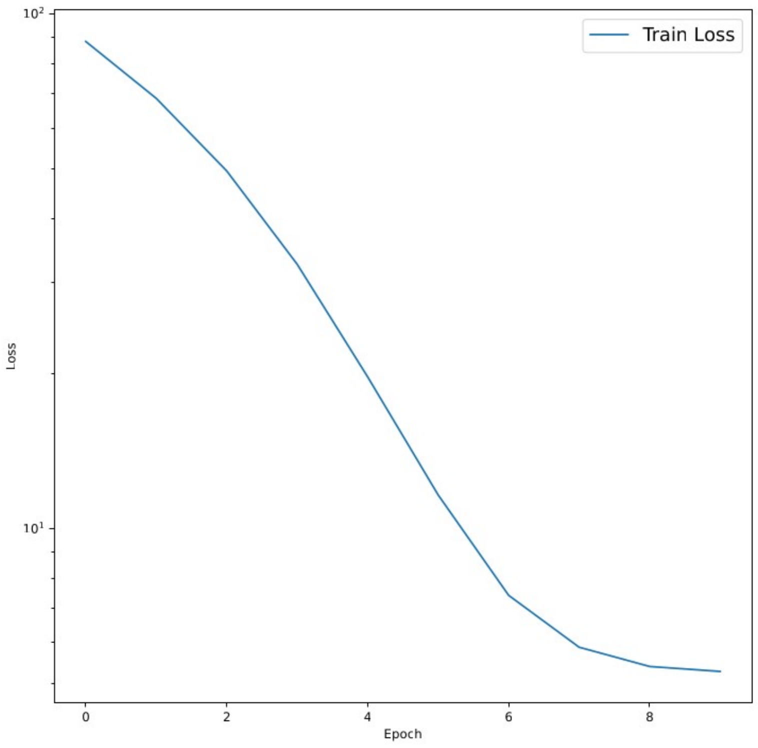
$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

My results

(4,2,1)   epochs = 10, batch_size = 10, loss='MeanSquaredError'

	y_true	y_pred	diffsquare
0	19.2770	20.875713	2.555884
1	10.2068	6.609452	12.940911
2	13.4036	11.171407	4.982687
3	16.9928	18.148956	1.336697
4	18.9150	20.083755	1.365989
5	16.3164	15.959281	0.127534
6	17.0892	17.902021	0.660679
7	17.7760	18.831499	1.114078
8	17.8712	18.611347	0.547818
9	12.1943	9.930314	5.125632

The total difference between expectation and prediction, the MSE = 3.0758



Assignment and practice

1. Use TOF, height, distance to predict angle (instead of velocity)

2. Use `projectile_input_10000.txt`
It has 10,000 examples. See if the training converges faster (less epochs, different batch_size). See if you can try a more complicated network, now that you have more training data. Test it on `projectile_input_test_100.txt`

3. Now use `projectile_input2_10000.txt` (and `projectile_input2_test_100.txt`)
It has 10,000 examples. Here the range of input velocities and angles is expanded. Do you need a more complicated network for this, to get same performance?

4. Use `projectile_input3_100k.txt` (and `projectile_input3_test_10k.txt`)
It has 100,000 examples. Here the variables are smeared… i.e. the TOF, height distance are smeared from their true value by anything from 5 to 15% … thus there isn't a mere analytic relation, but an additional stochastic noise component.
See if the problem is harder (i.e. worse performance, or requires more training, or more complex network needed etc.)