ML models for tabular datasets

1 Comparison between some models

1.1 Random forest: is an ensemble learning method for classification and regression. It constructs a multitude of decision trees at training time. For classification, the output is the one selected by the majority of trees and for regression, it outputs the average of trees. → They usually outperform decision trees.

The advantages of random forests over decision trees:

- Since they're consisted of many decision trees, they output a more accurate results compared to single trees.
- This also reduces overfitting
- Robust to outliers
- Work well on non-linear data

However, on the other hand, decision trees are easier to interpret and visualize. And it takes less time training them.

1.2 XGBoost: a decison_tree_based ensemble machine learning algorithm that uses gradient boosting framework. It provides parallel tree boosting and is the leading machine learning library for regression, classification and ranking problems.

The term "boosting" comes from the idea of improving a single weak model (here, it's decisio trees). In gradient boosting, the process of additively generating weak models is formalized as a gradient descent algorithm and it sets targeted outcomes for the next model in an effort to minimize errors.

GBDTs iteratively train an ensemble of shallow decision trees, with each iteration using the error residuals of the previous model to fit the next model. The final output is the weighted average of all trees.

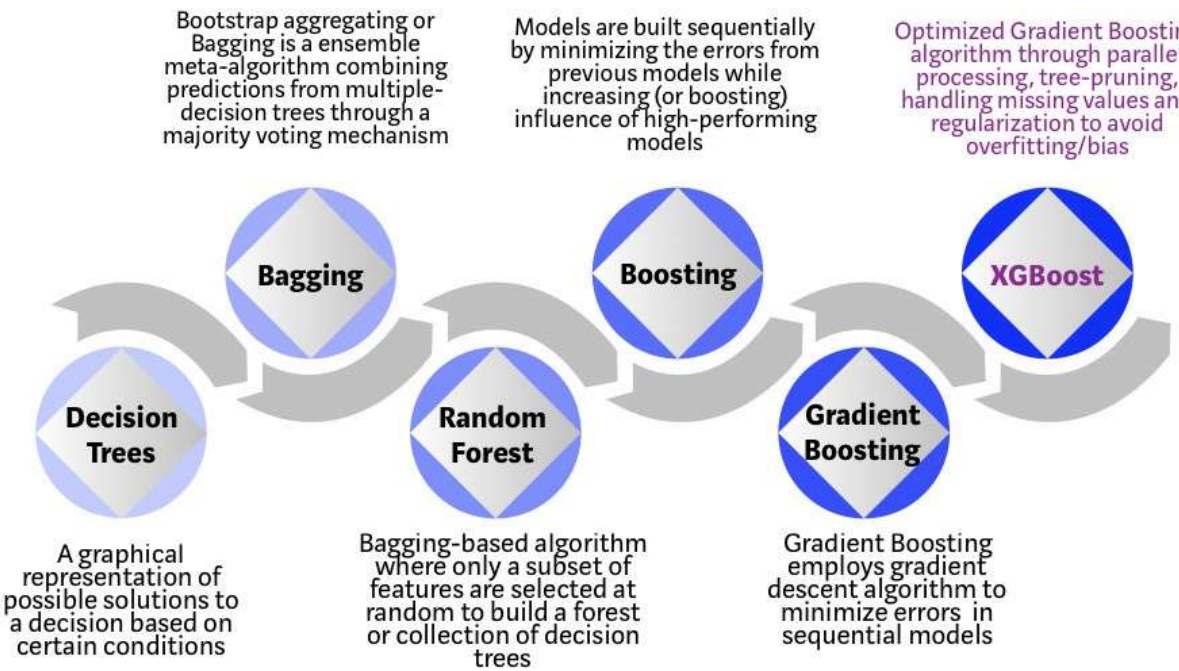
Random forest bagging minimizes the variance and overfitting. GBDT boosting minimizes the bias and underfitting

XGBoost is a scalable and highly accurate implementation of gradient boosting.

It has a high computational speed since the trees are built in parallel, instead of sequentialy.

Advantages of using XGBoost over:

- Random forests: • prunes the tree with a score called "similarity score" before entering into the actual modeling purposes • is a good option for unbalaced data • gives more importance to functional space when reducing the cost of a model whereas Random forest gives more preferences to hyperparameters. • more preferable in poisson regression and rank regression.
  • has larger accuracy and less prone to overfitting
- Ada boost: • higher speed and accuracy • more flexible • the complex observations are computed by large residues left on the previous iteration to increase the performance.
  • Can use different types of loss function.
- Gradient boosting: • more regularized (uses advanced reg) • higher performance since the training is fast and parallelized • preferable in the case of having large training samples or when there is a mixture of categorical and numeric features.



2. Fitting the model

2.1 How Random Forest fits the data?

Step 1: Select K random features

step2: Individual decision trees are constructed for each sample.

step3: Voting will take place by averaging the decision tree.

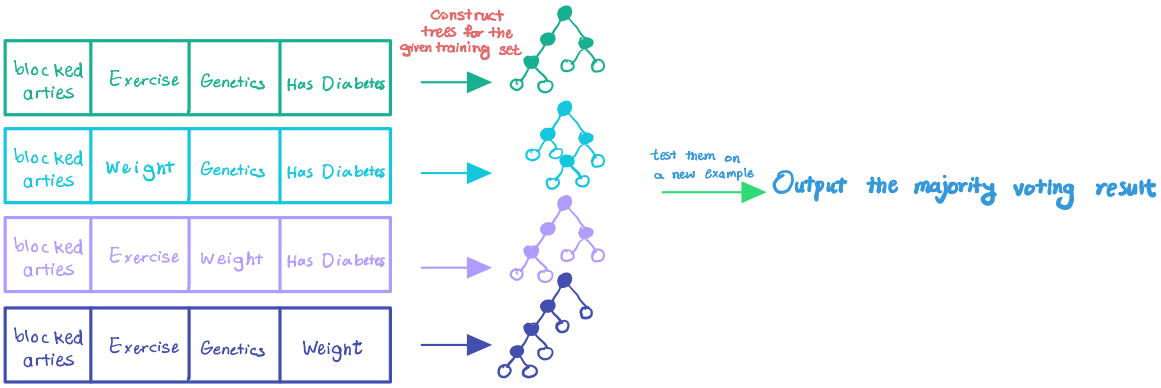
step4: Finally, select the most voted prediction result, as the final output.

(Or if it's a regression problem, output the average value)

To randomize the feature choice, we can pick a random subset of K features (from n feature in total) where K<n and allow the algorithm to only choose from that subset of features.

Blocked Arteries	Weight	Exercise	Genetics	Has Diabetes
No	210	Yes	Yes	No
No	125	No	No	No
Yes	180	Yes	Yes	Yes
Yes	167	No	Yes	Yes

In the case of the example above, we can constrct some trees of a subset of 4 feature and then for a test data, we can calculate the output on each of them and do majority voting. The process might look something like this:



- How Ada boost fits the data?

Initialize k : the number of rounds

Initialize D: the training data set

Initialize $w_1(i) = 1/n \quad i = 1, \ldots, n$

for $r = 1$ to k do:

For all i: $w_r(i) := w_r(i) / \sum_i w_r(i)$ [normalize weight]

$h_r := \text{Fit Weak Learner}(D, w_r)$

$\varepsilon_r := \sum_i w_r(i)\, 1(h_r(i) \neq y_i)$

if $\varepsilon_r > 1 - 1/C^{\#\text{classes}}$ then stop

$\alpha_r := \frac{1}{2} \log[(1-\varepsilon_r)/\varepsilon_r] + \log(C-1)$ [α_r indicates the importance of that classifier]

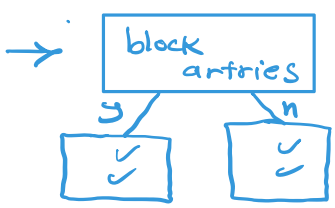
$$w_{r+1}(i) := w_r(i) \times \begin{cases} e^{-\alpha_r} & \text{if } h_r(x^{[i]}) = y^{[i]} \\ e^{\alpha_r} & \text{if } h_r(x^{[i]}) \neq y^{[i]} \end{cases}$$

Predict $h_{ens}(x) = \arg\max_j \sum_r^k \alpha_r 1[h_r(x) = j]$

- It usually make use of stumps and some of them play a more important role than others in predicting the output value → they learn from the mistakes of the previous ones.

In the first step all sample get a same weight → in this example → $\frac{1}{4}$

To build the first stump, we check how well each of these features can classify the samples. → here: blocked cortries: (we could calculate the gini or entropy for each to determine this)



all the samples are classified correctly → Total Error = 0 (based on the weights of the missclassified ones)

→ amount of Say $= \frac{1}{2} \log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right)$ → here it's a large positive value

If had a sample getting classified incorrectly, we will increase its weigh using formula: Sample weight $\times e^{\text{amount of say}}$ ⇒ Then decrease weights

of the correctly classified ones. and normalize them → Make the second stump using the new weights obtained and weighted Gini Index.

2.2 Usually Random Forest handles missing data in two ways:

1) Drop the data points containing missing values (not recommended)

2) Fill them with median (regression) of respective columns or mode (classification)

→ we can also pre-compute them by normalizing features and then constructg a distance metric and filling them with ave of k-nearest neighbors.

Also another idea is to make an initial guess for the missing values and the gradually refine the guess → we can choose the most common one

And also median.

We determine which samples are similar to the one with missing data.

To determine similarity → build a random forest and run all the data through it can see which ones end up in the same leafs → use a proximity matrix → a row for each sample & column

we put a one if they ended up in the same node Repeat this for other trees and fill the matrix → update it. → divide values by the number of trees

Then use proximity to make better guesses for the missing data using the frequency of each of them.. (Use weighted frequency for features) → use the one with higher frequency → we repeat until it converges.

3 Gradient boost

Step 1: construct a base tree (just the root node)

Step 2: Build next tree based on errors of the previous tree

Step 3: Combine tree from step 1 with trees from step 2. Go back to step 2.

Algorithm 1 Gradient Boosting

1: Initialize T: the number of trees for gradient boosting rounds
2: Initialize $\mathcal{D}$: the training dataset, $\{\langle \mathbf{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{n}$
3: Choose $L(y^{(i)}, h(\mathbf{x}^{(i)}))$, a differentiable loss function
4: **Step 1**: Initialize model $h_0(\mathbf{x}) = \underset{\hat{y}}{\operatorname{argmin}} \sum_{i=1}^{n} L(y^{(i)}, \hat{y})$ [root node]
5: **Step 2**:
6: **for** t=1 to T **do**
7: **A.** Compute pseudo residual $r_{i,t} = -\left[\frac{\partial L(y^{(i)}, h(\mathbf{x}^{(i)}))}{\partial h(\mathbf{x}^{(i)})}\right]_{h(\mathbf{x})=h_{t-1}(\mathbf{x})}$, for $i = 1$ to n
8: **B.** Fit tree to r_{i_t} values, and create terminal nodes $R_{j,t}$ for $j = 1, ..., J_t$.
9: **C.**
10: **for** j=1 to J_t **do**
11: $\hat{y}_{j,t} = \underset{\hat{y}}{\operatorname{argmin}} \sum_{\mathbf{x}^{(i)} \in R_{i,j}} L(y^{(i)}, h_{t-1}(\mathbf{x}^{(i)}) + \hat{y})$
12: **D.** Update $h_t(\mathbf{x}) = h_{t-1}(\mathbf{x}) + \alpha \sum_{j=1}^{J_t} \hat{y}_{j,t} \mathbb{I}(\mathbf{x} \in R_{j,t})$
13: **Step 3**: Return $h_t(\mathbf{x})$

The initial tree is basically the data values in each node: $h_0 \rightarrow$ 

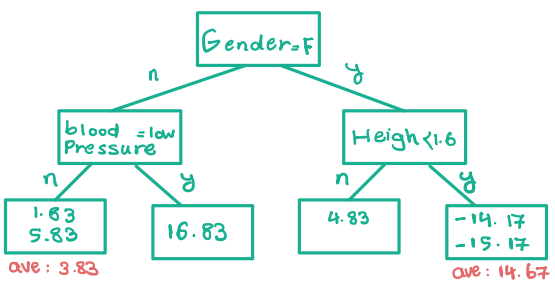
$\arg\min_{\hat{y}} \sum (y_i - \hat{y})^2 \rightarrow \hat{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$

$\rightarrow \hat{y}_1 = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} = 71.17$

Now compute (pseudo) residuals $r_1 = y_1 - \hat{y}_1$

4.83
16.83
-15.17
-14.17
1.83
5.83

Then creat a tree based on $x_1, \_, x_m$ to fit the residuals



$\Rightarrow h_t(x) = h_{t-1}(x) + \alpha \hat{y}_t$:

4.83			4.83		71.65		4.35
16.83			16.83		72.83		15.17
-15.17	+	α (0.1)	-14.67	=	69.7	residual: $\Rightarrow$	-13.7
-14.17			-14.67		69.7		-12.7
1.83			3.83		71.5		1.4
5.83			3.83		71.5		5.4

Next we can fit a tree to this residuals in the exact similar way we can continue this procedure for more rounds and fit trees

Intro to NNs

4 Descriptive Questions

4.1 Exploding Gradient problem: it describes a situation in training of neural networks where the gradients used to update the weights grow exponentially. This results in very large updates to NN weights. This way network becomes unstable and backpropagation can't update the weights properly. This happens through exponential growth by repeatedly multiplying gradients through the network layers that have values larger than 1 or less than -1. This usually happens in networks with large number of layers and these large numbers getting multiplied repeatedly.

How to prevent it?

- Re-design the network model: having fewer layers or smaller batch size,...
- Use long short-term memory networks
- Gradient Clipping: to limit the size of gradients in training by checking them against a threshold value and clipped or set to that threshold if it exceeds that
- Use weight regularization: apply a penalty to networks loss function for large weight values.
- Use activation functions such as sigmoid, tanh or ReLU that have small gradient
- Reduce the learning rate
- Use batch Norm

4.2

Pros of adding more layers to NN: usually accuracy goes up especially when the function we want to estimate is complex, having more layers helps to approximate the ideal function more properly.

Cons of adding more layers to NN: makes the model more complex - longer training time - we need more data for training - Gradient Exploding / Vanishing.

May result in overfitting

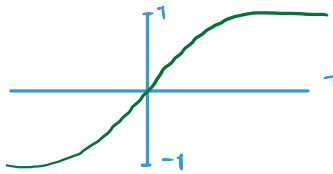
4.3

why shuffling in the first step of Stochastic Gradient Descent?

In the case of having ordered data, SGD will have problems finding the global optima and may end up finding local minima or face slower convergence if the data doesn't get shuffled. Hence, this is likely to Morever, it helps training convergence faster and prevents the model from learning the order of training Mini-batch methods that rely on SGD which rely on the randomness to find a minimum. Shuffling makes the gradients more variable which can help convergence by increasing the chance of hitting a good direction Morever, initial shuffling prevents having a cyclic behaviour when computing gradient.

4.4

sigmoid activation function: $S(x) = \frac{1}{1 + e^{-x}}$



The larger the input, closer to 1.

The output of sigmoid saturates for a large positive or negative number. This way, the gradient in these regions becomes almost zero. During BP, this local gradient is multiplied with the gradient of this gate's output. Hence, having a small local gradient, will kill the gradient and the network won't be able to learn properly → Problem of Vanishing Gradients.

This problem can be solved using ReLU activation function in hidden layers → $f(x) = \max(0, x)$

In the negative region, the gradients are also zero and again the weights don't get updated However, in the positive side, the gradients are one.

4.5

All the outputs will be assigne to class 1:

$$\text{ReLU}(z) = \begin{cases} 0 & z \leq 0 \\ z & z > 0 \end{cases} \rightarrow \sigma(\text{ReLU}(z)) = \begin{cases} \sigma(0) = \frac{1}{1+1} = \frac{1}{2} & z \leq 0 \\ \sigma(z) > \frac{1}{2} & z > 0 \end{cases}$$

σ: 

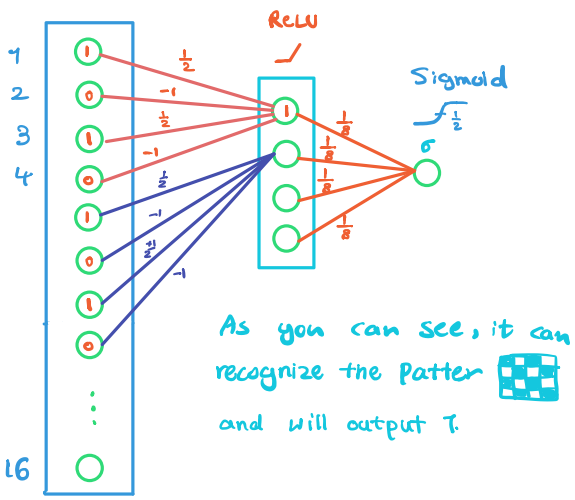
⇒ Network's output is always $\geq \frac{1}{2}$

4.6

A NN without activation functions, is just a linear model. There will be no point in using multi-lager structure. The output will be a linear function of inputs. Hence, network won't be able to develop complex structure and functions that would not be possible by using a simple linear model.

5 Intuitive Questions

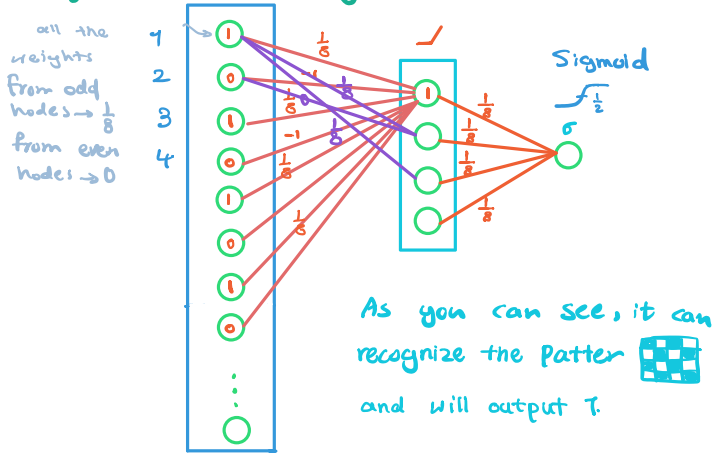
We can see the 4x4 image as a input vector of size 16. which are the inputs of the neural network. Also, for the output layer we only need one node which indicates wether it matches the pattern or not.



we will assign the weights to be $\frac{1}{2}$ and -1 as you can see. Also note that every 4 nodes have a non-zero weighted edge to a node in the second layer. If any of the even numbered nodes were white then, the corresponding node in the second layer will be a negative value and hence, the node in 3rd layer will have a value less than $1/2 \rightarrow$ it won't get matched. Also if any of the odd numbered nodes were black, again the neuron in 3rd layer will have a value less than $\frac{1}{2}$.

We could also have the following structure if we wanted to have non-zero edges:

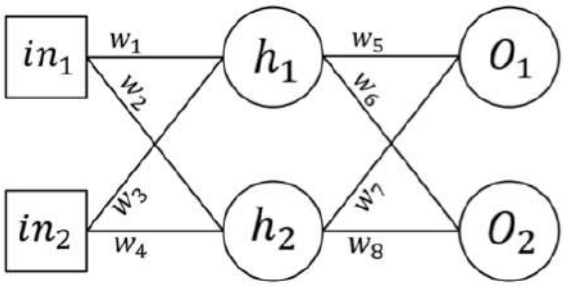
with all the weighs from first layer to second one being $\frac{-1}{8}$ and -1.



5.2 MLP is good for simple image classification, CNN is good for complicated image classification. MLPs use one perceptron for each input (pixel) and the amount of weights becomes unmanageable for large images. It includes too many parameters because it's fully-connected hence, we will have difficulties training on large images and overfitting can also happen. Moreover, they are not translation varients of an object in the image.

The main problem is that spatial information is lost when the image is flattened to a vector

Neural Network. (The activation function is sigmoid)



$w_i = i \times 0.1$

$in_1 = 0.1$

$in_2 = 0.5$

$b_{h1} = 0.25$

$b_{h2} = 0.25$

$b_{o2} = 0.35$

$b_{o2} = 0.35$

$lr = 1$

$t_1 = 0.05$

$t_2 = 0.95$

$E = \frac{1}{2}\sum(t_1 - o_2)^2$

Forward pass:

the value of h_1's input

$z_{h1} = w_1 in_1 + w_3 in_2 + b_{h1} = 0.1 \times 0.1 + 0.3 \times 0.5 + 0.25 = 0.01 + 0.15 + 0.25 = 0.41$

$a_{h1} = \sigma(z_{h1}) = \frac{1}{1 + e^{-0.41}} = 0.601$

$z_{h2} = w_2 in_1 + w_4 in_2 + b_{h2} = 0.2 \times 0.1 + 0.4 \times 0.5 + 0.25 = 0.02 + 0.2 + 0.25 = 0.47$

$a_{h2} = \sigma(h_2) = \frac{1}{1 + e^{-0.47}} = 0.615$

$z_{o1} = w_5 \cdot a_{h1} + w_7 a_{h2} + b_{o1} = 0.5 \times 0.601 + 0.7 \times 0.615 + 0.35 = 1.081$

$a_{o1} = \sigma(z_{o1}) = 0.746$

$z_{o2} = w_6 a_{h1} + w_8 a_{h2} + b_{o2} = 0.6 \times 0.601 + 0.8 \times 0.615 + 0.35 = 1.203$

$a_{o2} = \sigma(z_{o2}) = 0.769$

$E = \frac{1}{2}\left[(t_1 - a_{o1})^2 + (t_2 - a_{o2})^2\right] = \frac{1}{2}\left[(0.05 - 0.746)^2 + (0.9 - 0.769)^2\right] = 0.259$

Back Propagation Step:

We know that we update the weights as $w_i := w_i - lr \frac{\partial E}{\partial w_i}$

Hence, we should calculate the gradient of E with respect to w_is using chain rule.

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial a_{o_1}} \times \frac{\partial a_{o_1}}{\partial z_{o_1}} \times \frac{\partial z_{o_1}}{\partial w_5} = (a_{o_1} - t_1) \times (a_{o_1} \times (1 - a_{o_1})) \times a_{h_1} = 0.079$$

$$\frac{\partial E}{\partial w_7} = \frac{\partial E}{\partial a_{o_1}} \times \frac{\partial a_{o_1}}{\partial z_{o_1}} \times \frac{\partial z_{o_1}}{\partial w_7} = (a_{o_1} - t_1) \times (a_{o_1} \times (1 - a_{o_1})) \times a_{h_2} = 0.081$$

$$\frac{\partial E}{\partial w_6} = \frac{\partial E}{\partial a_{o_2}} \times \frac{\partial a_{o_2}}{\partial z_{o_2}} \times \frac{\partial z_{o_2}}{\partial w_6} = (a_{o_2} - t_2) \times (a_{o_2} \times (1 - a_{o_2})) \times a_{h_1} = -0.019$$

$$\frac{\partial E}{\partial w_8} = \frac{\partial E}{\partial a_{o_2}} \times \frac{\partial a_{o_2}}{\partial z_{o_2}} \times \frac{\partial z_{o_2}}{\partial w_8} = (a_{o_2} - t_2) \times (a_{o_2} \times (1 - a_{o_2})) \times a_{h_2} = -0.019$$

Similarly:

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial a_{o_1}} \times \frac{\partial a_{o_1}}{\partial z_{o_1}} \times \frac{\partial z_{o_1}}{\partial a_{h_1}} \times \frac{\partial a_{h_1}}{\partial z_{h_1}} \times \frac{\partial z_{h_1}}{\partial w_1} + \frac{\partial E}{\partial a_{o_2}} \times \frac{\partial a_{o_2}}{\partial z_{o_2}} \times \frac{\partial z_{o_2}}{\partial a_{h_1}} \times \frac{\partial a_{h_1}}{\partial z_{h_1}} \times \frac{\partial z_{h_1}}{\partial w_1} = 0.001$$

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial a_{o_1}} \times \frac{\partial a_{o_1}}{\partial z_{o_1}} \times \frac{\partial z_{o_1}}{\partial a_n} \times \frac{\partial a_{h_2}}{\partial z_{h_2}} \times \frac{\partial z_{h_2}}{\partial w_2} + \frac{\partial E}{\partial a_{o_2}} \times \frac{\partial a_{o_2}}{\partial z_{o_2}} \times \frac{\partial z_{o_2}}{\partial a_{h_2}} \times \frac{\partial a_{h_2}}{\partial z_{h_2}} \times \frac{\partial z_{h_2}}{\partial w_2} = 0.0015$$

$$\frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial a_{o_1}} \times \frac{\partial a_{o_1}}{\partial z_{o_1}} \times \frac{\partial z_{o_1}}{\partial a_{h_1}} \times \frac{\partial a_{h_1}}{\partial z_{h_1}} \times \frac{\partial z_{h_1}}{\partial w_3} + \frac{\partial E}{\partial a_{o_2}} \times \frac{\partial a_{o_2}}{\partial z_{o_2}} \times \frac{\partial z_{o_2}}{\partial a_{h_1}} \times \frac{\partial a_{h_1}}{\partial z_{h_1}} \times \frac{\partial z_{h_1}}{\partial w_3} = 0.005$$

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial a_{o_1}} \times \frac{\partial a_{o_1}}{\partial z_{o_1}} \times \frac{\partial z_{o_1}}{\partial a_{h_2}} \times \frac{\partial a_{h_2}}{\partial z_{h_2}} \times \frac{\partial z_{h_2}}{\partial w_2} + \frac{\partial E}{\partial a_{o_2}} \times \frac{\partial a_{o_2}}{\partial z_{o_2}} \times \frac{\partial z_{o_2}}{\partial a_{h_2}} \times \frac{\partial a_{h_2}}{\partial z_{h_2}} \times \frac{\partial z_{h_2}}{\partial w_4} = 0.007$$

$$w_1 := w_1 - 1 \times \frac{\partial E}{\partial w_1} = 0.988$$

$$w_2 := w_2 - 1 \times \frac{\partial E}{\partial w_2} = 0.198$$

$$w_3 := w_3 - 1 \times \frac{\partial E}{\partial w_3} = 0.294$$

$$w_4 := w_4 - 1 \times \frac{\partial E}{\partial w_4} = 0.392$$

$$w_5 := w_5 - 1 \times \frac{\partial E}{\partial w_5} = 0.420$$

$$w_6 := w_6 - 1 \times \frac{\partial E}{\partial w_6} = 0.619$$

$$w_7 := w_7 - 1 \times \frac{\partial E}{\partial w_7} = 0.618$$

$$w_8 := w_8 - 1 \times \frac{\partial E}{\partial w_8} = 0.819$$

We've calculated all these partial values and we can substitude them and calculate them easily.