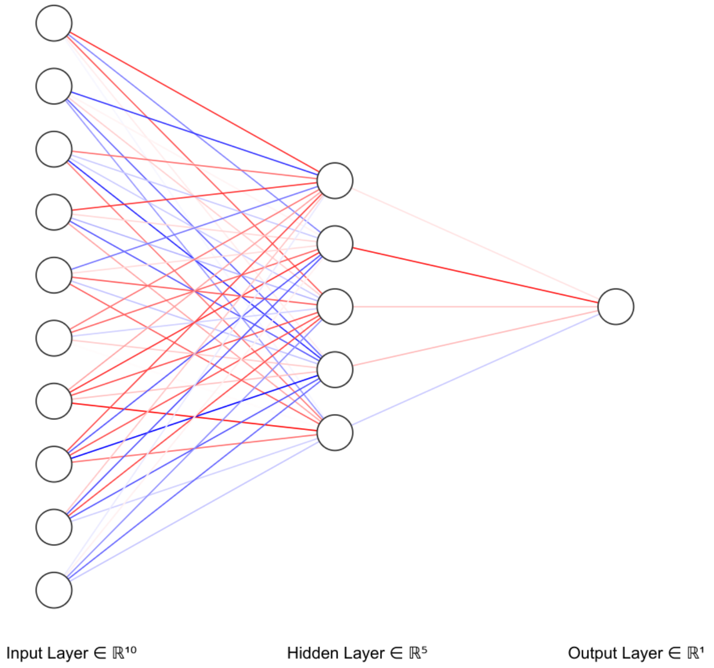
Neural Network from scratch


Input Layer $\in \mathbb{R}^{10}$
Hidden Layer $\in \mathbb{R}^{5}$
Output Layer $\in \mathbb{R}^{1}$

Dot product of Vector and Matrix

$$c_k = \sum^{i} a_i b_{ik}$$

In [1]:

```
def vec_dot_mat(a, b):
    c = []
    for k in range(len(b[0])):
        s = 0
        for i in range(len(a)):
            s += a[i] * b[i][k]
        c.append(s)
    return c
```

Random matrix

In [2]:

```
from random import uniform

def rand_matrix(i, j):
    return [[uniform(-1, 1) for _ in range(j)] for _ in range(i)]
```

Let us take,

- Input matrix X of size (10, 10)
- True weight1 matrix W_{t1} of size (10, 5)
- True bias1 b_{t1}
- True weight2 matrix W_{t2} of size (5, 1)
- True bias1 b_{t2}

In [3]:

```
x = rand_matrix(10, 10)
wt1 = rand_matrix(10, 5)
bt1 = uniform(-1, 1)
```

```
wt2 = rand_matrix(5, 1)
bt2 = uniform(-1, 1)
```

We have to find out below weights and biases

In [4]:
```
w1 = rand_matrix(10, 5)
b1 = uniform(-1, 1)
w2 = rand_matrix(5, 1)
b2 = uniform(-1, 1)
```

$$\mathrm{forward}_l = \sum^{k} \left[\left[\sum^{j} x_j . w1_{jk} + b1 \right]_k . w2_{kl} \right] + b2$$

In [5]:
```
def forward(x, w1, w2, b1, b2):
    z1 = [i+b1 for i in vec_dot_mat(x, w1)]
    z2 = [i+b2 for i in vec_dot_mat(z1, w2)]
    return z2
```

In [6]:
```
y = [forward(i, wt1, wt2, bt1, bt2) for i in x]
```

Modification of weight

In [7]:
```
def mod_w(w, index, h):
    w[index[0]][index[1]] += h
    return w
```

$$\mathrm{loss} = \frac{1}{N} \sum^{l} \left\{ y_l - \sum^{k} \left[\left[\sum^{j} x_j . w1_{jk} + b1 \right]_k . w2_{kl} \right] + b2 \right\}^2$$

In [8]:
```
def loss(x, y, w1, w2, b1, b2):
    l = 0
    yp = forward(x, w1, w2, b1, b2)
    for i in range(len(y)):
        l += (y[i] - yp[i]) ** 2
    l /= len(y)
    return l
```

$$\mathrm{overall\backslash\_loss} = \frac{1}{N^2} \sum^{i} \sum^{l} \left\{ y_{il} - \sum^{k} \left[\left[\sum^{j} x_{ij} . w1_{jk} + b1 \right]_k . w2_{kl} \right] + b2 \right\}^2$$

In [9]:
```
def overall_loss(x, y, w1, w2, b1, b2):
    yp = [forward(i, w1, w2, b1, b2) for i in x]
    l = 0
    for i in range(len(y)):
        for j in range(len(y[0])):
            l += (y[i][j] - yp[i][j]) ** 2
    l /= len(y) * len(y[0])
    return l
```

$$\mathrm{grad}_w = \frac{\mathrm{loss}(mod_w(w, (i, j), h)) - \mathrm{loss}(mod_w(w, (i, j), -h))}{2h}$$

$$\mathrm{grad}_b = \frac{\mathrm{loss}(b + h) - \mathrm{loss}(b - h)}{2h}$$

$$w1 \leftarrow w1 - lr.\triangle w1$$

$$w2 \leftarrow w2 - lr.\triangle w2$$

$$b1 \leftarrow b1 - lr.\triangle b1$$

$$b2 \leftarrow b2 - lr.\triangle b2$$

In [10]:

```
def grad(x, y, w1, w2, b1, b2, h, lr):
    dw1 = []
    for i in range(len(w1)):
        temp = []
        for j in range(len(w1[0])):
            temp.append((loss(x, y, mod_w(w1, (i, j), h), w2, b1, b2) \
                - loss(x, y, mod_w(w1, (i, j), -h), w2, b1, b2)) / (2 * h))
        dw1.append(temp)

    dw2 = []
    for i in range(len(w2)):
        temp = []
        for j in range(len(w2[0])):
            temp.append((loss(x, y, w1, mod_w(w2, (i, j), h), b1, b2) \
                - loss(x, y, w1,
                       mod_w(w2, (i, j), -h), b1, b2)) / (2 * h))
        dw2.append(temp)

    db1 = (loss(x, y, w1, w2, b1+h, b2) \
          - loss(x, y, w1, w2, b1-h, b2)) / (2 * h)
    db2 = (loss(x, y, w1, w2, b1, b2+h) \
          - loss(x, y, w1, w2, b1, b2-h)) / (2 * h)

    for i in range(len(w1)):
        for j in range(len(w1[0])):
            w1[i][j] -= lr * dw1[i][j]

    for i in range(len(w2)):
        for j in range(len(w2[0])):
            w2[i][j] -= lr * dw2[i][j]

    b1 -= lr * db1
    b2 -= lr * db2

    return w1, w2, b1, b2
```

In [11]:

```
epochs = 1000
lr = 0.01
h = 0.001
opt_w_b = (w1, w2, b1, b2)
lowest_loss = overall_loss(x, y, w1, w2, b1, b2)
print('Inital loss: {}'.format(lowest_loss))
for _ in range(epochs):
    for x_row, y_row in zip(x,y):
        w1, w2, b1, b2 = grad(x_row, y_row, w1, w2, b1, b2, h, lr)
    l = overall_loss(x, y, w1, w2, b1, b2)
    if l < lowest_loss:
        lowest_loss = l
        opt_w_b = (w1, w2, b1, b2)
print('Lowest loss: {}'.format(lowest_loss))
print('Final loss: {}'.format(overall_loss(x, y, w1, w2, b1, b2)))
```

```
Inital loss: 13.28357350279802
Lowest loss: 7.2511179192521535e-06
Final loss: 7.2511179192521535e-06
```

Rust code

```
use rand::Rng;

fn main() {
    let mut rng = rand::thread_rng();

    fn vec_dot_mat(a:&Vec<f64>, b:&Vec<Vec<f64>>) -> Vec<f64> {
        let mut c = vec![];
        for k in 0..b[0].len() {
            let mut s = 0.0;
            for j in 0..a.len() {
                s += a[j] * b[j][k];
            }
            c.push(s);
        }
        c
    }

    fn rand_matrix(i:usize, j:usize) -> Vec<Vec<f64>> {
        let mut rng = rand::thread_rng();
        let mut m = vec![vec![0.0;j];i];
        for a in 0..i {
            for b in 0..j {
                m[a][b] = rng.gen_range(-1.0..=1.0);
            }
        }
        m
    }

    let x = rand_matrix(10, 10);
    let wt1 = rand_matrix(10, 5);
    let bt1 = rng.gen_range(-1.0..=1.0);
    let wt2 = rand_matrix(5, 1);
    let bt2 = rng.gen_range(-1.0..=1.0);

    let mut w1 = rand_matrix(10, 5);
    let mut b1 = rng.gen_range(-1.0..=1.0);
    let mut w2 = rand_matrix(5, 1);
    let mut b2 = rng.gen_range(-1.0..=1.0);

    fn forward(x:&Vec<f64>, w1:&Vec<Vec<f64>>, w2:&Vec<Vec<f64>>, b1:&f64,
b2:&f64) -> Vec<f64> {
        let z1:Vec<f64> = vec_dot_mat(x,
w1).iter().map(|i|i+b1).collect();
        let z2 = vec_dot_mat(&z1, w2).iter().map(|i|i+b2).collect();
        z2
    }

    let y:Vec<Vec<f64>> = x.iter().map(|i| forward(i, &wt1, &wt2, &bt1,
&bt2)).collect();

    fn mod_w(w:&Vec<Vec<f64>>, index:(usize, usize), h:f64) ->
Vec<Vec<f64>>{
        let mut w1 = w.clone();
        w1[index.0][index.1] += h;
        w1
    }
    fn loss(x:&Vec<f64>, y:&Vec<f64>, w1:&Vec<Vec<f64>>,
w2:&Vec<Vec<f64>>, b1:&f64, b2:&f64) -> f64 {
        let mut l = 0.0;
        let yp = forward(x, w1, w2, b1, b2);
        for i in 0..y.len() {
            l += (y[i] - yp[i]).powi(2);
        }
        l /= y.len() as f64;
        l
    }

    fn overall_loss(x:&Vec<Vec<f64>>, y:&Vec<Vec<f64>>, w1:&Vec<Vec<f64>>,
w2:&Vec<Vec<f64>>, b1:&f64, b2:&f64) -> f64 {
        let mut yp = vec![];
        for i in x {
            let temp = forward(i, w1, w2, b1, b2);
            yp.push(temp);
        }
        let mut l = 0.0;
        for i in 0..y.len() {
            for j in 0..y[0].len() {
                l += (y[i][j] - yp[i][j]).powi(2);
            }
        }
        l /= (y.len() * y[0].len()) as f64;
        l
    }

    fn grad(x:&Vec<f64>, y:&Vec<f64>, w1:&Vec<Vec<f64>>,
w2:&Vec<Vec<f64>>, b1:&f64, b2:&f64, h:f64) -> (Vec<Vec<f64>>,
Vec<Vec<f64>>, f64, f64) {
        let mut dw1 = vec![];
        for i in 0..w1.len() {
            let mut temp = vec![];
            for j in 0..w1[0].len() {
                temp.push((loss(x, y, &mod_w(w1, (i, j), h), w2, b1, b2) -
loss(x, y, &mod_w(w1, (i, j), -h), w2, b1, b2)) / (2.0 * h));
            }
            dw1.push(temp);
        }

        let mut dw2 = vec![];
        for i in 0..w2.len() {
            let mut temp = vec![];
            for j in 0..w2[0].len() {
                temp.push((loss(x, y, w1, &mod_w(w2, (i, j), h), b1, b2) -
loss(x, y, w1, &mod_w(w2, (i, j), -h), b1, b2)) / (2.0 * h));
            }
            dw2.push(temp);
        }

        let db1 = (loss(x, y, w1, w2, &(b1+h), b2) - loss(x, y, w1, w2, &
(b1-h), b2)) / (2.0 * h);
        let db2 = (loss(x, y, w1, w2, b1, &(b2+h)) - loss(x, y, w1, w2,
b1, &(b2-h))) / (2.0 * h);

        (dw1, dw2, db1, db2)
    }

    let epochs = 1000;
    let lr = 0.01;
    let h = 0.001;
    let mut opt_w_b = (w1.clone(), w2.clone(), b1.clone(), b2.clone());
    let mut lowest_loss = overall_loss(&x, &y, &w1, &w2, &b1, &b2);
    println!("Inital loss: {}", lowest_loss);
    for _ in 0..epochs {
        for index in 0..x.len() {
            let (dw1, dw2, db1, db2) = grad(&x[index], &y[index], &w1,
&w2, &b1, &b2, h);
            for i in 0..w1.len() {
                for j in 0..w1[0].len() {
                    w1[i][j] -= lr * dw1[i][j];
                }
            }

            for i in 0..w2.len() {
                for j in 0..w2[0].len() {
                    w2[i][j] -= lr * dw2[i][j];
                }
            }

            b1 -= lr * db1;
            b2 -= lr * db2;
        }
        let l = overall_loss(&x, &y, &w1, &w2, &b1, &b2);
        if l < lowest_loss {
            lowest_loss = l;
            opt_w_b = (w1.clone(), w2.clone(), b1.clone(), b2.clone());
        }
    }
    println!("Lowest loss: {}", lowest_loss);
    println!("Final loss: {}", overall_loss(&x, &y, &w1, &w2, &b1, &b2));
    // println!("{:?}", opt_w_b);
}
```

output:

```
    Inital loss: 14.449090232922213
    Lowest loss: 0.000005498838681749966
    Final loss: 0.000005498838681749966
```