1.2 Adaptative learning rate: ADAM

Summary

Why artificial neural network works?....1
Artificial neural network structure....1
How neural network predicts: forward propagation....5
How neural network learns: back propagation....7
Cost/error function....8
Gradient vector....9
Math behind network learning....11
Using the matrix's power to calculate the gradient vector....16
Bibliography....18

What is ADAM?

ADAM is an algorithm that adjusts the values of the error weights and biases dinamically, which allows finding a better local minimum in cost function and avoiding being stuck in bad local minimums.

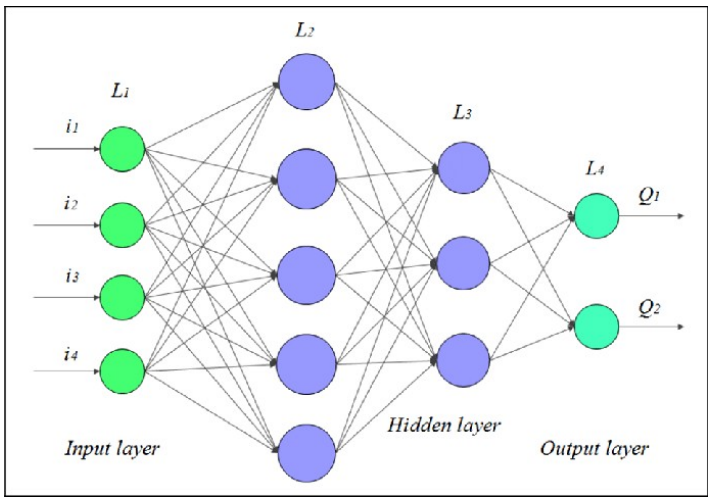
This algorithm consider this parameters:

- Learning_rate: the same value seen in previous chapters
- B1: a value between [0.0,1.0)
- B2: a value between [0.0,1.0)
- Epsilon: a value near 0, normally 10^{-8}

How does it work?

He first initialize two same matrices with same shape for each weight matrices but all elements 0, and two same vectors with same shape for each bias vectors but all elements 0. One of those matrix/vector is known as momentum, and the other one as velocity.

Let's create the lists for this network:



Our optimization list will be:

Optimization = [

Momentum,

Velocity

]

Momentum = [

Momentum_weights,

Momentum_biases

]

Momentum_weights =[

[matrix 4x5 all elements 0],

[matrix 5x3 all elements 0],

[matrix 3x2 all elements 0]

]

Momentum_biases = [

[matrix 1x5 all elements 0],

[matrix 1x3 all elements 0],

[matrix 1x2 all elements 0]

]

Velocity = [

Velocity_weights,

Velocity_biases

]

Velocity_weights =[

[matrix 4x5 all elements 0],

[matrix 5x3 all elements 0],

[matrix 3x2 all elements 0]

]

Velocity_biases = [

[matrix 1x5 all elements 0],

[matrix 1x3 all elements 0],

[matrix 1x2 all elements 0]

]

We can see there's a bunch of matrices, but all of them are needed

Next, we calculate gradient vector with weights and biases errors, represented as GV(dw,db), where dw are all matrices with weight error values, and db al vectors with bias error values

Now, consider t-1 the moment before computing the gradient vector, and t the moment after computing it. We first are going to replace all values in Momentum weights with this operation. This is the formula:

$$\mathbf{m}_{t+1} = \beta_1 \mathbf{m}_t + (1 - \beta_1)\mathbf{g}(\mathbf{w}_t)$$

Now we are going to show a pseudo-code:

```
for i in range(length( dw )):
    for r in range(number_rows( dw[i] )):
        for c in range(number_cols( dw[i] )):
            Momentum_weights[i][r][c] = B1 * Momentum_weights[i][r][c] + (1 – B1) * dw[i][r][c]
```

The same for biases:

```
for i in range(length( db )):
    for j in range(length( db[i] )):
        Momentum_biases[i][j] = B1 * Momentum_biases[i][j] + (1 – B1) * dw[i][j]
```

Now let's calculate the velocity values. This is the formula:

$$\mathbf{v}_{t+1} = \beta_2 \mathbf{v}_t + (1 - \beta_2)\mathbf{g}(\mathbf{w}_t)^2$$

This will be the pseudo-code:

```
for i in range(length( dw )):
    for r in range(number_rows( dw[i] )):
        for c in range(number_cols( dw[i] )):
            Velocity_weights[i][r][c] = B2 * Velocity_weights[i][r][c] + (1 – B2) * (dw[i][r][c])^2
```

The same for biases:

```
for i in range(length( db )):
    for j in range(length( db[i] )):
        Velocity_biases[i][j] = B2 * Velocity_biases[i][j] + (1 – B2) * (dw[i][j])^2
```

The next step is to adjust all this values respect of B1 and B2. Theoretically this will be like:

$$\hat{\mathbf{m}}_{t+1} = \mathbf{m}_{t+1} / \left(1 - \beta_1{}^{t+1}\right)$$

$$\hat{\mathbf{v}}_{t+1} = \mathbf{v}_{t+1} / \left(1 - \beta_2{}^{t+1}\right)$$

However we are going to do this when substracting the gradient vector to the network, in order to reduce the number of needed matrices and vectors.

Finally, the last step would be:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{\varepsilon + \sqrt{\hat{\mathbf{v}}_{t+1}}} \hat{\mathbf{m}}_{t+1}$$

In this formula is only shown the weight adjust, but the same is done for bias values. You only need to change w to b.

Let’s consider W all network weights and B all network biases. The pseudo-code to adjust all this values is:

```
for i in range(length(W)):
    for r in range(number_rows(W[i])):
        for c in range(number_cols(W[i])):
```

```
            fit_momentum = Momentum_weights[i][r][c] / (1 – B1)
            fit_velocity = Velocity_weights[i][r][c] / (1 – B2)
            W[i][r][c] -= (Learning_rate * fit_momentum) / ( Epsilon + math.sqrt( fit_velocity ) )
for i in range(length( B )):
    for j in range(length( B[j] )):
        fit_momentum = Momentum_biases[i][j] / (1 – B1)
        fit_velocity = Velocity_biases[i][j] / (1 – B2)
        B[i][j] -= (Learning_rate * fit_momentum) / ( Epsilon + math.sqrt( fit_velocity ) )
```

This should be enough to implement Adam in our neural network

Bibliography

Nice web: https://towardsdatascience.com/how-to-implement-an-adam-optimizer-from-scratch-76e7b217f1cc

Original article: https://arxiv.org/pdf/1412.6980.pdf

“Supossed to be” good video: https://www.youtube.com/watch?v=JXQT_vxqwIs

Good video in my opinion: https://www.youtube.com/watch?v=NE88eqLngkg